

СУ “Св. Климент Охридски”
Факултет по Математика и Информатика

Програмиране на Java

Материали за упражненията по „Увод в програмирането”, „Структури от данни и програмиране” и „Структури от данни и обектно-ориентирано програмиране” за специалностите „Приложна математика” и „Компютърни науки”

Съставил Емилия Живкова

zivkova@fmi.uni-sofia.bg

2012 г.

Предговор

В съответствие с новите практики е използвана програмната система Moodle за разширяване на обучението по време на упражненията по дисциплините „Увод в програмирането”, „Структури от данни и програмиране”, „Структури от данни и обектно-ориентирано програмиране” за специалностите „Приложна математика” и „Компютърни науки” с възможността за използване на Интернет и с цел - повишаване на ефективността на обучението и подобряване на организацията на процеса на обучение. Авторът е участвал при разработката на следните курсове:

1. *Курс „Структури от данни и програмиране”*. Упражнения по дисциплината „Структури от данни и програмиране” на специалност „Приложна математика” 1 курс, седмичен хорариум 4+2 (2 часа семинарни), лектор доц. д-р Д. Биров, летен семестър на учебната 2008/2009 г. – участие с материали за 3 група

2. *Курс „Структури от данни и обектно-ориентирано програмиране”*. Упражнения по дисциплината „Структури от данни и обектно-ориентирано програмиране” на специалност “Компютърни науки” 1 курс, седмичен хорариум 4+4 (2 часа семинарни и 2 часа лабораторни), лектор доц. д-р Д. Биров, летен семестър на учебната 2008/2009 г. – участие с материали за 3 и 4 групи

3. *Курс „Увод в Програмирането”*. Упражнения по дисциплината „Увод в програмирането” на специалност “Компютърни науки” 1 курс, седмичен хорариум 4+4 (2 часа семинарни и 2 часа лабораторни), лектор доц. д-р Д. Биров, зимен семестър на учебната 2009/2010 г. – участие с материали за 3 и 4 групи

4. *Курс „Структури от данни и обектно-ориентирано програмиране”*. Упражнения по дисциплината „Структури от данни и обектно-ориентирано програмиране” на специалност “Компютърни науки” 1 курс, седмичен хорариум 4+4 (2 часа семинарни и 2 часа лабораторни), лектор доц. д-р Д. Биров, летен семестър на учебната 2009/2010 г. – участие с материали за 3 и 4 групи

5. *Курс „Увод в Програмирането”*. Упражнения по дисциплината „Увод в програмирането” на специалност “Компютърни науки” 1 курс, седмичен хорариум 4+4 (2 часа семинарни и 2 часа лабораторни), лектор доц. д-р Д. Биров, зимен семестър на учебната 2010/2011 г. – участие с материали за 4, 5 и 6 групи

Материалите за посочените групи са организирани, съгласно тематичните планове на дисциплините и включват:

- Текст за семинарното занятие (файл в PDF формат) за всяка седмица, съдържащ примерни програми или програмни фрагменти, илюстриращи съответната тема, както и задачи по темата, които се обсъждат и решават
- Пълна или частична реализация на Java (файлове в TXT формат) на разглежданите задачи по темата
- Програмни шаблони на Java (файлове в TXT формат) за самостоятелна работа по време на лабораторните занятия или у дома
- Задания за лабораторните занятия или домашна работа (файлове в DOC формат) - задачи по темата, като файловете с реализациите на Java се изпращат на асистента

При съставянето на този текст са използвани разработените от автора материали. В него са включени примерни програми, задачи, решавани по време на упражненията или предоставяни на студентите за самостоятелна работа, както и техните реализации на езика Java. Авторът се е опитал да подпомогне читателя чрез включване на уводна част, кратко представяне на някои от темите, диаграми, таблици и др., при което си е позволил и по-подробно да представи типовете в Java.

E. Живкова

Съдържание

Увод	1
Тема 1. Програмиране на аритметични изрази	9
Тема 2. Програмиране на цикли и разклонения	37
Тема 3. Процедурно програмиране	52
Тема 4. Програмиране с използване на масиви	66
Тема 5. Обработка на текстове	84
Тема 6. Програмиране с използване на рекурсия	93
Тема 7. Обектно-ориентирано програмиране	103
Тема 8. Стек	143
Тема 9. Дек	155
Тема 10. Вектор	163
Тема 11. Итератор	173
Тема 12. Опашка	175
Тема 13. Списък	182
Тема 14. Двоично дърво	202
Тема 15. Таблица	222
Тема 16. Речник	236
Тема 17. Префиксно дърво	242
Тема 18. Множество	257
Тема 19. Граф	260
Литература	277



Решаване на задача с използване на компютър

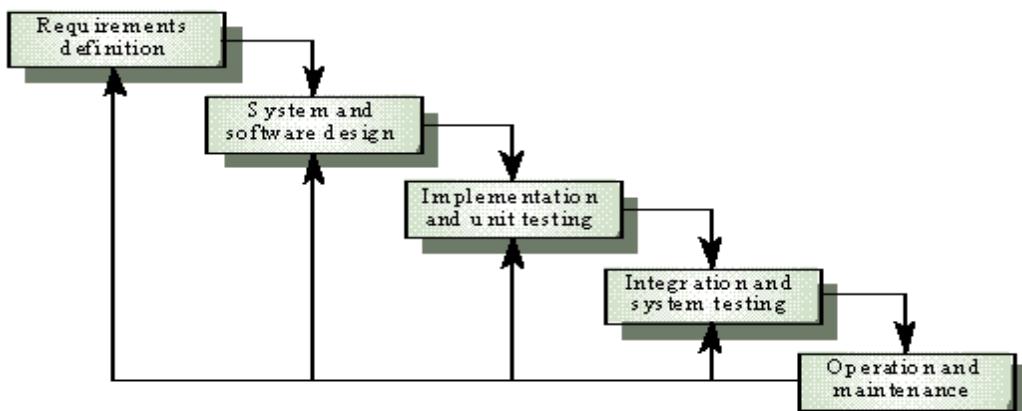
Основни дейности:

- *Анализ.* Съставяне на ясна, точна и без двусмислици формулировка на задачата. Изясняване какво е необходимо за нейното решаване, като се определят: входните данни (параметри), обработките, които трябва да се извършат над тях и крайният резултат (изходните данни)
- *Проектиране.* Описание на основните компоненти, от които се състои решението на задачата и отношенията между тях
- *Реализация.* Представяне на решението във вид на **програма**, използвайки език за програмиране
- *Тестване.* Изпълнение на програмата от компютър с различни входни данни с цел установяване, че тя работи така както се очаква
- *Модификация на програмата*

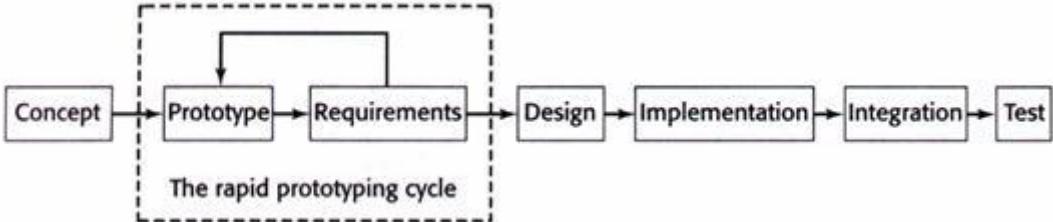
Основни подходи:

- *Процедурно програмиране* – представяне на решението на задачата чрез структури от данни, определени от типове данни и функции (процедури, подпрограми, методи) за тяхната обработка. Процесът на изпълнение на програмата се състои в изпълнение на функциите, чрез *извикване на функциите*. Когато функция А иска да се изпълни функция В, то А извика В, при което двете функции предават данни помежду си
- *Обектно-ориентирано програмиране* – организиране на структурите от данни и методите за тяхната обработка в *класове*, като всеки клас обединява структури от данни и съответните методи за обработка. Процесът на изпълнение на програмата се състои в създаване на *екземпляри на класовете* (обекти), които взаимодействат помежду си чрез изпращане на *съобщения* (извикване на методи в Java). Когато обект А иска обект В да изпълни някой от методите на класа, на който В е екземпляр, то А изпраща съобщение (извика метода) на В
- *Обобщено програмиране* – разработка на програмни компоненти (generics, templates) с общо предназначение, без ограничение за типа на данните, с възможност за многократно използване

Процес на разработка на програма: Представени са два модела на процеса на разработка на програмна система – виж <http://www.robabdul.com/Data-Management-System-Software-Development-Cycle.asp>, които може да се следват и при разработка на програма



Водопаден модел (Royce, 1970)



Модел на прототипите (Е. М. Bennatan, 2000)

Подготовка на програма за изпълнение от компютър

Обработката на програма, водеща до нейното изпълнение от компютър, включва:

1. *Въвеждане на програмата и съхранението ѝ в изходен файл* чрез използване на програма **текстов редактор**

2. *Транслиране (компиляция или интерпретация) и изпълнение на програмата*

2.1. Компиляция

2.1.1. *Транслиране на програмата на машинен език* от програма **компилатор**, като новият вид на програмата се съхранява в **обектен файл**

2.1.2. *Обработка на обектния файл*, наречена **свързване**, от програма **свързващ редактор**, като резултатът е **изпълним файл**

2.1.3. *Зареждане на изпълнимия файл в оперативната памет на компютъра и изпълнението му*, което се осигурува от програма **операционна система**

2.2. Интерпретация

2.2.1. *Транслиране на програмата на интерпретируем език* от програма **компилатор**, като новият вид на програмата се съхранява във файл

2.2.2. *Интерпретация на програмата* от програма **интерпретатор**, която чете интерпретируемия вид на програмата ред по ред и го изпълнява

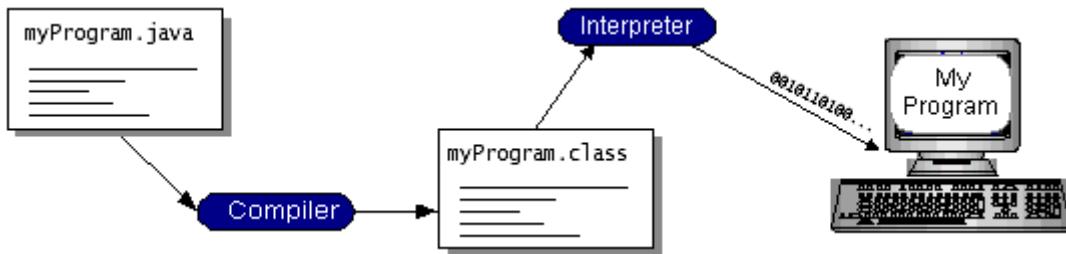
Модулно програмиране: Организиране на компонентите на програмата в *модули*. Всеки модул е обединение на ресурси, достъпът до които е ясно и точно специфициран, и се състои от две части: *спецификация* („видима“ част), описваща *интерфейса* с предоставяните ресурси и *реализация* („скрита“ част) на самите ресурси. Всеки модул се транслира еднократно, като обектният код се съхранява в *библиотека* и може да се използва многоократно. Свързването на модул с програма, която го използва, се осъществява на етапа на свързване на програмата.

Програмиране на езика Java

При съставянето на текста са използвани материали и свободно разпространявани програмни средства, достъпни в Internet. Следващата таблица съдържа адреси на някои от страниците и кратко описание на тяхното съдържание.

Адрес	Съдържание
http://docs.oracle.com/javase/tutorial/	Ръководство за използване на езика Java
The Java Language Specification, Third Edition	Спецификация на езика Java
http://java.sun.com/docs/books/jvms/	Спецификация на виртуалната машина на езика Java
http://docs.oracle.com/javase/6/docs/api/	Платформа Java SE 6 – документация
http://www.java.com/en/	Платформа Java – програмни средства
http://www.eclipse.org/webtools	Среда за програмиране Eclipse

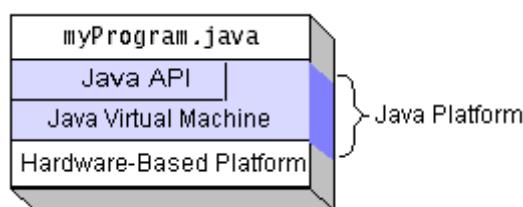
Обработката на програма на езика Java е представена на диаграмата



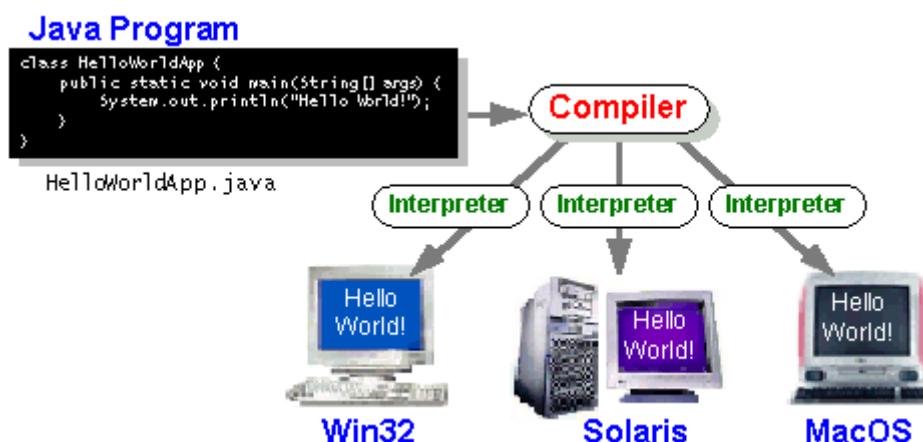
Програмата се транслира на междинен език, наречен *Java bytecodes*, който зависи от *платформата*.

Платформата представлява хардуерната или софтуерната среда, в която се изпълнява програмата. Платформата на Java е само софтуерна и се състои от две компоненти:

- Java VM
- Java API (Application Programming Interface) – колекция от софтуерни компоненти, които могат да се използват при програмирането



След това всяка инструкция на този език се изпълнява от съответния за платформата интерпретатор. Програмата се транслира еднократно и интерпретира при всяко изпълнение. На този език може да се гледа като на машинен език на *виртуална машина на Java* (Java VM) и всеки интерпретатор се явява негова реализация. Това дава възможност програмата да се транслира на всяка платформа, която има компилатор и да бъде изпълнена на всяка реализация на Java VM, както е показано на следната диаграма



1. Структура на прости програми на Java

Традиционният метод за решаване на прости задачи се състои от следните стъпки:

1. Определяне на входните данни, обработките, които трябва да се извършват над тях и на изходните данни
2. Определяне на **алгоритъм** за решаване на задачата
3. Представяне на алгоритъма във вид на **програма**, използвайки език за програмиране
4. Подготовка на програмата за изпълнение от компютър
5. Изпълнение на програмата от компютър с различни входни данни с цел установяване, че тя работи така както се очаква

Алгоритъмът за решаване на поставената задача се представя като *програма* на Java, при което обработките се оформят като *методи*. Всеки метод принадлежи на *клас*, а всеки клас – на *пакет*, като името на пакета се задава в инструкцията **package**. В метод на клас може да се извика всеки от:

- Разрешените за използване методи на същия клас
- Разрешените за използване методи от класовете на същия пакет
- Разрешените за използване методи от класовете на други пакети, като за всеки от използвани пакети се включва инструкцията **import**

Има няколко вида програми на Java – приложения, аплети и др., като за всеки вид програма е специфицирано, кои са нейните задължителни методи и как тя се изпълнява. Програма-приложение задължително има метод с име **main**, от който започва нейното изпълнение.

Програма, състояща се само от метод с име **main**, има следния вид:

```
[package Име_на_пакет; ]  
[Използвани_пакети]  
  
public class Име_на_клас {  
    public static void main(String[] arg)  
        Тяло_на_метода  
}
```

и тя се съхранява в текстов файл с име **Име_на_клас.java**.

2. Програмиране на Java за операционната система Microsoft Windows

За **Microsoft Windows** са необходими:

- Текстов редактор
- Платформа на Java

Примерен сценарий с използване на текстовият редактор **Notepad** и платформа на **Java**:

1. Въвеждане на програмата
 - 1.1. Стаптирайте **Notepad** и въведете текста на програма, която извежда Hello, World!

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

- 1.2. Съхранете програмата във файл с име **HelloWorld.java** в директорията **C:\JavaPrograms** с команда **File/Save As**
- 1.3. Излезте от текстовия редактор
2. Транслиране на програмата
 - 2.1. Изпълнете команда **Start/MS-DOS Promt** за Windows 95/98 или приложението **Command Promt** за по-късни версии на Windows
 - 2.2. Ако е необходимо, променете текущото устройство с команда **C:** и /или текущата директория с команда **cd C:\JavaPrograms**
 - 2.3. Транслирайте програмата с команда **javac HelloWorld.java**. Ако има съобщения за грешки, преминете на т. 1 за тяхното отстраняване и повторна транслация. Ако транслацията е успешна, в директорията се появява файл с име **HelloWorld.class**
 - 2.4. Излезте с команда **exit** или изпълнете програмата, съгласно т. 3
3. Изпълнение на програмата
 - 3.1. Изпълнете програмата (един или повече пъти) с команда **java HelloWorld** и в прозорец на экрана ще видите Hello, World!

3.2. Излезте с командата **exit**

За програмите в текста са използвани:

- Средата за програмиране **Eclipse**
- Платформата **Java(TM) 6 Update 20**

Използването на тези средства включва:

- Създавне и обработка на програмата **HelloWorld.java** в **Eclipse**

```
package Intro;

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

- Изпълнение на програмата **HelloWorld** в **Eclipse**

Hello, World!

Представяне на числата

В основата на решението на много задача лежи *аритметиката*. Изпълнението на четирите основни аритметични действия: *събиране, изважддане, умножение и деление* зависи от това как са представени числата над които те се изпълняват.

1. Реални числа

Нека $R \geq 2$ е цяло число, $Digits = \{0, 1, \dots, R-1\}$ и $M > 0$ е цяло число. Представянето на реално число с фиксирана точка в позиционна бройна система с основа R с точност M цифри се определя по правилото:

$$\pm(d_N d_{N-1} \dots d_1 d_0 . d_1 d_2 \dots d_M)_R = \\ = \pm(d_N R^N + d_{N-1} R^{N-1} + \dots + d_1 R + d_0 R^0 + d_1 R^{-1} + d_2 R^{-2} + \dots + d_M R^{-M}) = D \quad (1)$$

където $d_i \in Digits$ за всяко $i = -M, \dots, N$.

Примери за позиционни бройни системи (ПБС):

- Десетична: $R=10$ и $Digits = \{0, 1, \dots, 9\}$

$$366 = 3 \times 10^2 + 6 \times 10^1 + 6 \times 10^0$$

$$13.375 = 1 \times 10^1 + 3 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2} + 5 \times 10^{-3}$$

- Двоична: $R=2$ и $Digits = \{0, 1\}$

$$(101101110)_2 = 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 +$$

$$+ 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 366$$

$$(1101.011)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} =$$

$$= 8 + 4 + 1 + 0.25 + 0.125 = 13.375$$

- Осмична: $R=8$ и $Digits = \{0, 1, \dots, 8\}$

$$(15.3)_8 = 1 \times 8 + 5 \times 8^0 + 3 \times 8^{-1} = 8 + 5 + 0.375 = 13.375$$

- Шестнадесетична: $R=16$ и $Digits = \{0, \dots, 9, A, B, C, D, E, F\}$

$$(D.6)_{16} = D \times 16^0 + 6 \times 16^{-1} = 13 + 0.375 = 13.375$$

$$(16E)_{16} = 1 \times 16^2 + 6 \times 16^1 + E \times 16^0 = 1 \times 16^2 + 6 \times 16^1 + 14 \times 16^0 = 366$$

2. Аритметични действия с реални числа в една и съща ПБС

Аритметичните действия между числа, записани в ПБС с дадена основа, се определят от правилото (1).

Събирането, умножението и изваждането се основават на таблиците за събиране, умножение и изваждане на едноцифрени числа и на правилата за събиране, умножение и изваждане на многоцифрени числа, познати от десетичните числа.

По отношение на действията целочислено деление (/) и деление по модул (%) в сила е следното: ако A и B са естествени числа, то съществуват цели числа $P \geq 0$ и $0 \leq Q < B$, такива че $A = B \times P + Q$ и те са единствени. Числото P се нарича *частно* и $P = A / B$, а Q се нарича *остатък* и $Q = A \% B$.

Сравнението на числа в една и съща ПБС се извършва аналогично на това при десетичните числа.

3. Преобразуване от една ПБС в друга

Ще представим методите за преобразуване от ПБС с основа R в добре познатата ни десетична ПБС и обратно.

3.1. Преобразуване на неотрицателно реално число с фиксирана точка от ПБС с основа R в десетично число. Това преобразуване се състои в пресмятане стойността на формулата (1). Броят на операциите, необходими за получаване на числото D, може да се намали, прилагайки *правилото на Хорнер*, поотделно за цялата и дробната част:

$$D = (((d_N R + d_{N-1}) \times R + d_{N-2}) \times R + \dots + d_1) \times R + d_0 + ((\dots((d_{-M} \times R^{-1} + d_{-M+1}) \times R^{-1} + d_{-M+2}) \times R^{-1} + \dots + d_{-1}) \times R^{-1}) \quad (2)$$

Пример:

$$\begin{aligned} (1101.011)_2 &= (((1 \times 2 + 1) \times 2 + 0) \times 2 + 1) + (((1 \times 2^{-1} + 1) \times 2^{-1} + 0) \times 2^{-1}) = \\ &= ((3 \times 2 + 0) \times 2 + 1) + ((1.5 \times 2^{-1} + 0) \times 2^{-1}) = (6 \times 2 + 1) + ((0.75 + 0) \times 2^{-1}) = \\ &= 13 + (0.75 \times 2^{-1}) = 13 + 0.375 = 13.375 \end{aligned}$$

Алгоритъм ConversionToDecimal: Дадено е числото $D_R = (d_N d_{N-1} \dots d_1 d_0. d_{-1} d_{-2} \dots d_{-M})_R$. Като резултат се получава неговото десетично представяне D.

1. Получаване на цялата част на D в променливата iPart
 - 1.1. iPart=0
 - 1.2. k = N
 - 1.3. Ако $k \geq 0$, переход на т. 1.4. Иначе – переход на т.2
 - 1.4. iPart = iPart $\times R + d_k$
 - 1.5. k = k - 1
 - 1.6. Преход на т. 1.3
2. Получаване на дробната част на D в променливата fPart
 - 2.1. fPart = 0
 - 2.2. k = -M
 - 2.3. Ако $k \leq -1$, переход на т. 2.4. Иначе – переход на т.3
 - 2.4. fPart = (fPart + d_k) / R
 - 2.5. k = k + 1
 - 2.6. Преход на т. 2.3
3. Получаване на числото D в променливата result
 - 3.1. result = iPart + fPart
4. Край на алгоритъма

3.2. Преобразуване на естествено число в число в ПБС с основа R. Цифрите на числото се получават чрез последователно деление, съгласно правилото (1).

Алгоритъм RepeatedDivision: Дадени са естественото число D и цялото число R ≥ 2 . Като резултат се получава $(d_N d_{N-1} \dots d_1 d_0)_R$ - представяне на числото D в ПБС с основа R.

1. $k = -1$
2. $k = k + 1$
3. $d_k = D \% R$
4. $D = D / R$
5. Ако $D \neq 0$, преход на т. 2. Иначе – край на алгоритъма

Пример: Да се намери представянето на числото D = 267 в ПБС с основа 16. Решението ще представим във вид на таблица:

k	$d_k = D \% 16$	$D = D / 16$
-1	неопределено	267
0	$d_0 = 11$	16
1	$d_1 = 0$	1
2	$d_2 = 1$	0

Следователно, $267 = (10B)_{16}$.

3.3. Преобразуване на положителна десетична дроб в дроб в ПБС с основа R и с точност M цифри. Цифрите на числото се получават чрез последователно умножение, съгласно правилото (1).

Алгоритъм RepeatedMultiplication: Дадени са десетичната дроб F, цялото число R ≥ 2 и цялото число M ≥ 1 . Като резултат се получава $(d_1 \dots d_M)_R$ - представяне на числото F в ПБС с основа R и с точност M цифри.

1. $k = 0$
2. $k = k - 1$
3. $prod = F \times R$
4. $d_k = INT(prod)$
5. $F = FRACT(prod)$
6. Ако $k \neq -M$, преход на т. 2. Иначе – край на алгоритъма

При описанието на алгоритъма са използвани функциите INT(P) и FRACT(P) за получаване съответно на цялата и дробната част на числото P. Например, INT(3.92)=3 и FRACT(3.92)=0.92.

Пример: Да се представи десетичната дроб F = 0.961 в ПБС с основа 8 и с точност M = 3 цифри. Решението ще представим във вид на таблица:

k	$prod = F \times 8$	$d_k = INT(prod)$	$F = FRACT(prod)$
0	неопределено	неопределено	неопределено
-1	7.688	$d_{-1} = 7$	0.688
-2	5.504	$d_{-2} = 5$	0.504
-3	4.032	$d_{-3} = 4$	0.032

Следователно, $0.961 = (0.754)_8$.

4. Реални числа с плаваща точка

Нека $R \geq 2$ е цяло число и $Digits = \{0, 1, \dots, R-1\}$. Представянето на реално число в плаващ формат в ПБС с основа R се определя по правилото: $\text{mantissa} \times R^{\text{exp}}$, където мантисата е реално число с фиксирана точка и експонентата exp - цяло число, са числа в ПБС с основа R .

Примери за аритметични действия с такива числа:

$$0.1652 \times 10^3 + 0.2100 \times 10^2 = 0.1652 \times 10^3 + 0.0210 \times 10^3 = \\ = (0.1652 + 0.0210) \times 10^3 = 0.1862 \times 10^3 \\ 0.5 \times 10^2 \times 0.6 \times 10^3 = (0.5 \times 0.6) \times 10^{2+3} = 0.30 \times 10^5$$



Задани за упражнение

Задача: Да се преобразуват в ПБС с основа 10 числата: $(22.2)_8$, $(1011)_2$, $(AD.4)_{16}$, $(-77)_8$.

Задача: Да се преобразуват следващите десетични числа в дадената ПБС:

1. Числото 72 в ПБС с основа 16
2. Числото 36 в ПБС с основа 8
3. Числото -24 в ПБС с основа 2
4. Числото 16 в ПБС с основа 16

Задача: Да се преобразуват следващите десетични числа в дадената ПБС:

1. Числото 7.2 в ПБС с основа 16
2. Числото 0.275 в ПБС с основа 8
3. Числото 0.5625 в ПБС с основа 2
4. Числото 0.5625 в ПБС с основа 16
5. Числото -23.1 в ПБС с основа 8



Програмиране на аритметични изрази

Езикът Java предоставя средства за програмиране на аритметични пресмятания. Данните, подлежащи на обработка, се представят като *константи* и *стойности на променливи* или се получават в резултат на *изпълнение на методи*. Аритметичните пресмятания се представят чрез *аритметични изрази* и изпълняват чрез *оператори*. Ще разгледаме накратко тези средства, като се надяваме с това да дадем отговор на въпроса как да програмираме такива пресмятания, използвайки знанията и опита в съставянето и пресмятането на математически формули, както и защо за някои данни резултатът от изпълнението на програмата не съвпада с очаквания.

От алгоритъм към програма на езика Java

От програма, която не прави нищо

```
package Arithmetic;

/**
 * @author Somebody
 *
 */
public class Example {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }
}
```

към програма, която преобразува температура F по Фаренхайт в температура C по Целзий.

Спецификация:

Предназначение на програмата: *Пресмята температурата C по Целзий*
 Параметър: *Температурата F по Фаренхайт*
 Резултат: *Температурата C по Целзий*

Алгоритъм: Дадена е температурата F по Фаренхайт. Като резултат се получава температурата C по Целзий.

$$1. \quad C = (F - 32) * 5 / 9$$

Реализация:

```
package Arithmetic;

/**
 * @author Somebody
 *
 */
public class Example {

    /**
     * Calculates C
     */
}
```

```
* @param F
* @param args
* @return C
*/
public static void main(String[] args) {
    double C, F;
    java.util.Scanner sc = new java.util.Scanner(System.in);

    F = sc.nextDouble();

    C = (F - 32) * 5 / 9;

    System.out.println(C);
}
```

1. **Деклариране на променливите.** Променливите С и F се създават при обработката на тяхната декларация:

```
double C, F;
```

2. **Задаване на стойности на променливите за входните данни.** Температурата по Фаренхайт е *параметър (входна данна)* на програмата и се въвежда от клавиатурата по време на изпълнение на програмата. Четенето на въведената стойност се извършва от *обект на клас за вход/изход*. В случая се създава обектът sc на класа **java.util.Scanner** - виж <http://java.sun.com/javase/6/docs/api/>, като се използва наличният обект java.lang.System.in:

```
java.util.Scanner sc = new java.util.Scanner(java.lang.System.in);
```

- За обекта се извиква методът nextDouble() за четене на въведеното реално число и на променливата F се присвоява прочетената стойност чрез *оператор за присвояване*:

```
F = sc.nextDouble();
```

3. **Обработка на входните данни.** На променливата С се присвоява стойността на *аритметичния израз* $(F - 32) * 5 / 9$ чрез *оператор за присвояване*:

```
C = (F - 32) * 5 / 9;
```

4. **Извеждане на резултата от обработката.** Стойността на променливата С е температурата по Целзий и се явява *изходна данна* на програмата. Тя се извежда на экрана от *обект на клас за вход/изход*. В случая се използва наличният обект System.out и за него се извиква някой от методите println(C) или print(C), който се изпълнява чрез *оператора*:

```
System.out.println(C);
```

Модификация:

```
package Arithmetic;

import java.util.Scanner;

/**
 * @author Somebody
 */
public class Example {
```

```

* Calculates C
* @param F
* @param args
* @return C
*/
public static void main(String[] args) {
    double C, F;
    Scanner sc = new Scanner(System.in);

    System.out.println("Reading...");
    System.out.print("    F = ");
    F = sc.nextDouble();

    C = (F - 32) * 5 / 9;

    System.out.println("Printing: ");
    System.out.print("    C = ");
    System.out.println(C);

    System.out.println("Done!");
}
}

```

Изпълнение на програмата:

```

Reading...
    F = 98
Printing:
    C = 36.66666666666664
Done!

```

Тестване: Изпълнение на програмата от компютър с различни входни данни с цел установяване, че тя работи така както се очаква.

Следващите програми илюстрират различни средства, които могат да се използват за четене на числа, въведени от клавиатурата:

1. *Използване на пакета ccj* - виж <http://www.horstmann.com/ccj.html>

```

package InputOutput;

import ccj.*;

public class UseCCJ {
    public static void main(String[] args) {
        System.out.println("Reading...");
        System.out.print("    N = ");
        int N = Console.in.readInt();
        System.out.print("    F = ");
        double F = Console.in.readDouble();

        System.out.println("Printing...");
        System.out.println("    N = " + N);
        System.out.println("    F = " + F);

        System.out.println("Done!");
    }
}

```

2. *Използване на класа java.util.Scanner*

```
package InputOutput;
```

```

import java.util.Scanner;

public class UseScanner {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Reading...");
        System.out.print("    N = ");
        int N = sc.nextInt();
        System.out.print("    F = ");
        double F = sc.nextDouble();

        System.out.println("Printing...");
        System.out.println("    N = " + N);
        System.out.println("    F = " + F);

        System.out.println("Done!");
    }
}

```

Задача: Да се състави програма, която по наличните монети от по 1, 2, 5, 10, 20 и 50 стотинки в портфейл определя колко лева и стотинки има в него.

Алгоритъм: Дадени са наличните монети c1, c2, c5, c10, c20, c50 от по 1, 2, 5, 10, 20 и 50 стотинки съответно. Като резултат се получава наличната сума leva лева и st стотинки.

1. total = c1 + c2 * 2 + c5 * 5 + c10 * 10 + c20 * 20 + c50 * 50
2. leva = total / 100
3. st = total % 100

Реализация: Програма **Example1_1**

```

package Arithmetic;

import java.util.Scanner;

public class Example1_1 {
    public static void main(String[] args) {
        int total, c1, c2, c5, c10, c20, c50, leva, st;
        Scanner sc = new Scanner(System.in);

        System.out.println("Reading...");
        System.out.print("    c1 = ");
        c1 = sc.nextInt();
        System.out.print("    c2 = ");
        c2 = sc.nextInt();
        System.out.print("    c5 = ");
        c5 = sc.nextInt();
        System.out.print("    c10 = ");
        c10 = sc.nextInt();
        System.out.print("    c20 = ");
        c20 = sc.nextInt();
        System.out.print("    c50 = ");
        c50 = sc.nextInt();

        total = c1 + c2 * 2 + c5 * 5 + c10 * 10 + c20 * 20 + c50 * 50;
        leva = total / 100;
        st = total % 100;

        System.out.println("Printing: ");
        System.out.print("    " + leva + " leva ");
        System.out.println("    " + st + " stotinki");
    }
}

```

```

        System.out.println("Done!");
    }
}

```

Задача: Да се състави програма, която определя периметъра P, лицето S и ъгъла α (срещу страната a) на триъгълник със страни a, b и c по формулите:

$$P = a+b+c, p = P/2 \quad \cos(\alpha/2) = \sqrt{p(p-a)/(bc)}$$

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \quad \alpha = 2\arccos(\sqrt{p(p-a)/(bc)})$$

Алгоритъм: Дадени са страните на триъгълник a, b и c. Като резултат се получават неговите периметър P, лице S и ъгъл alpha, които се определят съгласно дадените формули.

Реализация: Програма **Example1_2**

```

package Arithmetic;

import java.util.Scanner;

public class Example1_2 {

    public static void main(String[] args) {
        double a, b, c;
        Scanner sc = new Scanner(System.in);

        System.out.println("Reading...");
        System.out.print("      a = ");
        a = sc.nextDouble();
        System.out.print("      b = ");
        b = sc.nextDouble();
        System.out.print("      c = ");
        c = sc.nextDouble();

        double P, S, alpha, p;
        P = a + b + c;
        p = P / 2.0;
        S = Math.sqrt(p * (p - a) * (p - b) * (p - c));
        alpha = 2.0 * Math.acos(Math.sqrt(p * (p - a) / (b * c)));

        System.out.println("Printing: ");
        System.out.println("      S = " + S);
        System.out.println("      P = " + P);
        System.out.println("      alpha = " + alpha + " rad");

        System.out.println("Done!");
    }
}

```

За определяне на стойностите на функциите $\arccos(x)$ и корен квадратен от x за дадена стойност на x – реално число, се използват съответно методите **Math.acos(x)** и **Math.sqrt(x)** на класа **java.lang.Math**.

Програмата пресмята стойностите на променливите p, s и alpha за различни стойности на променливите a, b и c без да проверява дали те са страни на триъгълник, т.е. $a>0, b>0, c>0, a+b>c, a+c>b$ и $b+c>a$. Затова в някои случаи тя извежда страни, от гледна точка на поставената задача резултати.

Задача: Да се състави програма, която определя общия обем течност (в литри), намираща се в N бутилки по 2 литра и в M кутии, всяка по 12 унции.

1 унция (за течности) = 29.586 милилитра
1 галон = 3.785 литра
1 унция = 28.3495 грама
1 паунд = 453.6 грама
1 инч = 2.54 сантиметра
1 фут = 30.5 сантиметра
1 миля = 1.609 километра

Алгоритъм: Дадени са броят на бутилките по 2 литра N и броят на кутиите по 12 унции M. Като резултат се получава общият обем течност в тях result.

$$1. \text{ result} = N * 2 + M * 12 * 29.586 / 1000$$

Реализация: Програма Example1_3

```
package Arithmetic;

import java.util.Scanner;

public class Example1_3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Reading...");
        System.out.print("      Enter N > 0: ");
        int N = sc.nextInt();
        System.out.print("      Enter M > 0: ");
        int M = sc.nextInt();

        double result = N * 2 + M * 12 * 29.586 / 1000;

        System.out.println("Printing: ");
        System.out.println("      result = " + result);

        System.out.println("Done!");
    }
}
```

Задача: Време hh:mm може да се представи като цяло четирицифрене число hhmm, като hh е часът, а mm са минутите. Да се състави програма, която чете две времена $t_1 < t_2$ в този вид в рамките на едно дененощие и определя разликата между тях в часове и минути.

Алгоритъм: Дадени са две времена $t_1 < t_2$ в рамките на едно дененощие. Като резултат се получава разликата между тях в часове h и минути m.

1. Преобразуване на времето t_1 в минути m1
2. Преобразуване на времето t_2 в минути m2
3. Определяне на разликата между t_1 и t_2 в минути diff
4. Преобразуване на разликата diff в часове h и минути m

Реализация: Програма Example1_4

```
package Arithmetic;

import java.util.Scanner;

public class Example1_4 {
    public static void main(String[] args) {
        final int a = 100;
        final int b = 60;
```

```
Scanner sc = new Scanner(System.in);

System.out.println("Reading...");
System.out.print("    Enter the first time: ");
int t1 = sc.nextInt();
System.out.print("    Enter the second time: ");
int t2 = sc.nextInt();

int m1 = t1 / a * b + t1 % a;
int m2 = t2 / a * b + t2 % a;
int dif = m2 - m1;
int h = dif / b;
int m = dif % b;

System.out.println("Printing: ");
System.out.println("    " + h + " hours and " + m + " minutes");

System.out.println("Done!");
}
}
```

Изпълнение на програмата:

```
Enter the first time: 0510
Enter the second time: 1010
5 hours and 0 minutes
```

Примитивни числови типове, константи и променливи. Оператор за присвояване. Операция за преобразуване на типа, аритметични операции, операции за присвояване и аритметични изрази. Преобразуване на типовете

Типовете в Java са:

- Примитивни типове
 - Числови типове
 - Целочислени типове **byte, short, int, long** и **char**
 - Типове с плаваща точка **float** и **double**
 - Логически тип **boolean**
- Съставни типове
 - Масиви
 - Класове
 - Интерфейси

Всеки тип се характеризира с *множество от стойности* и *множество от операции и методи*, които могат да се прилагат над тези стойности.

В Java има следните видове константи:

- Целочислени константи
- Константи с плаваща точка
- Булеви константи
- Символни константи
- Низове
- Константа null

Всяка константа има тип, съгласно който се определя размерът на полето в паметта на компютъра за нейната стойност и тя не може да се променя по време на изпълнение на програмата.

В Java има следните видове променливи:

- Променливи на клас
- Променливи на екземпляр на клас
- Компоненти на масив
- Параметри на методи
- Параметри на конструктори
- Параметри на обработчици на изключения
- Локални променливи

Всяка променлива трябва явно да бъде декларирана, при което задължително се задават нейните *тип* и *име*. Типът определя множеството от допустими стойности на променливата. Името трябва да бъде:

- Коректен идентификатор
- Различно от всяка **ключова дума, true, false и null**
- Уникално в областта на действие на променливата

При обработката на декларацията на променлива в паметта на компютъра се отделя поле, в което се съхранява нейната стойност.

Областта на действие на променлива е частта от програмата, която се определя от мястото на нейната декларация и в която текущата ѝ стойност може да бъде използвана и променяна. Всяка променлива трябва да получи начална стойност преди да бъде използвана. На променливата може да се присвои стойност чрез *оператор за присвояване* от вида *Име_на_променлива = Стойност;*.

В програма може да се използват *именувани константи*. Декларацията на именувана константа съдържа ключовата дума *final* и стойност, която не може да се променя по време на изпълнение на програмата.

1. Целочислени типове byte, short, int и long

Стойностите на целочислените типове **byte, short, int** и **long** са цели двоични числа, представени в *допълнителен формат*. В този формат най-лявата цифра определя знака на числото – 0 за положителни числа и 1 за отрицателни. Представянето с дължина N на двоично число A се получава по следния начин:

1. Ако $A > 0$, то A се разширява до N-цифreno двоично число чрез добавяне на нули отляво
2. Ако $A < 0$, то:
 - 2.1. Числото $-A$ се разширява до N-цифreno двоично число
 - 2.2. Полученото число се инвертира, като всяка цифра 0 се заменя с 1, а всяка цифра 1 се заменя с 0
 - 2.3. Инвертираното число се събира с 1

Най-голямото цяло положително двоично число, което може да се представи по този начин е 0111...111 и то е двоичен запис на десетичното число $D_{max}=2^{N-1}-1$. Представянето 1000...000 определя най-малкото цяло отрицателно двоично число, което е $-1000...000$ и то е двоичен запис на десетичното число $D_{min}=-2^{N-1}$. Положителната и отрицателната нула имат едно и също представяне 0000...000.

N определя множество от двоични числа, такова че за всеки елемент A, различен от най-малкото число, числото $-A$ също принадлежи на множеството. В паметта на компютъра всяко число от множеството се съхранява в поле от N бита. Таблица 1 съдържа стойностите на N, D_{min} и D_{max} за всеки от целочислените типове.

	Byte	Short	Int	Long
N	8	16	32	64
D _{min}	-2 ⁷	-2 ¹⁵	-2 ³¹	-2 ⁶³
D _{max}	2 ⁷ -1	2 ¹⁵ -1	2 ³¹ -1	2 ⁶³ -1

Таблица 1. Параметри на целочислените типове

Стойностите на целочислените типове представят целите десетични числа:

- От -128 до 127, включително, за **byte**
- От -32768 до 32767, включително, за **short**
- От -2147483648 до 2147483647, включително, за **int**
- От -9223372036854775808 до 9223372036854775807, включително, за **long**

Двоичното число В, което представя десетично число А, се нарича *допълнителен код* на А и се получава от двоичното представяне на А по гореописания начин. Обратно, числото А се получава от В по следния начин: ако най-лявата цифра на В е 0, то $A \geq 0$ и В се явява двоичен запис на числото А. В противен случай $A < 0$ и:

1. От двоичното число В се изважда числото 1
2. Инвертира се полученото двоично число
3. Инвертираното число се превръща в десетично число С и $A = -C$

Операциите, които могат да се прилагат над тези стойности, са:

- Операции, чиито резултат е стойност на типа **int** или **long**
 - Операции за смяна на знака + и -
 - Операции за умножение *, деление / и деление по модул %
 - Операции за събиране + и изваждане -
 - Операция за увеличаване с 1 ++ (префиксна и постфиксна форма)
 - Операция за намаляване с 1 -- (префиксна и постфиксна форма)
 - Операции за изместване <<, >> и >>>
 - Операция за побитово инвертиране ~
 - Побитови операции конюнкция &, дизюнкция | и сума по модул две ^
 - Операции за присвояване =, *=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^= и |=
- Операции за сравнение ==, !=, <, <=, > и >=, чиито резултат е стойност на типа **boolean**
- Условна операция ?:
- Операция за преобразуване на целочислена стойност в стойност на указан числов тип
- Операция за конкатениране на низове +

При изпълнението на операциите +, - и * се изчисляват съответно сумата, разликата и произведението на двета целочислени операнда. Ако десният операнд на операциите / и % е различен от нула, се изчислява съответно частното и остатъкът при целочисленото деление на левия операнд на десния. В противен случай, изпълнението им завършва с активиране на изключение `ArithmeticException`. Таблица 2 съдържа резултатите от изпълнението на тези операции за различни стойности на операндите А и В от тип **int**.

A	B	A+B	A-B	A*B	A/B	A%B
5	2	7	3	10	2	1
5	-2	3	7	-10	-2	1
-5	2	-3	-7	-10	-2	-1
-5	-2	-7	-3	10	2	-1
-2147483648	-1	2147483647	-2147483647	-2147483648	-2147483648	0
1	0	1	1	0	ArithmeticException	ArithmeticException

Таблица 2. Резултати на аритметични целочислени операции

Резултатите от изпълнението на операциите за смяна на знака за различни стойности на операнда А от тип **int** са дадени в Таблица 3.

A	+A	-A
5	5	-5
-5	-5	5
-2147483648	-2147483648	-2147483648
0	0	0

Таблица 3. Резултати на целочислени операции за смяна на знака

Резултатите от изпълнението на операциите за присвояване, увеличаване и намаляване с 1 с операнди A=5 и B=2 от тип **int**, както и новата стойност на A, са дадени в Таблица 4.

	A+=B	A-=B	A*=B	A/=B	A%=B	A=B	A++	++A	A--	--A
Резултат	7	3	10	2	1	2	5	6	5	4
A	7	3	10	2	1	2	6	6	4	4

Таблица 4. Резултати на целочислени операции за присвояване, увеличаване и намаляване с 1

Класовете **Byte**, **Short**, **Integer**, **Long** и **Math** от пакета `java.lang` предлагат още възможности за обработка на целочислени стойности чрез декларирани в тях конструктори, методи и константи - виж <http://java.sun.com/javase/6/docs/api/>.

Програма Integers: Програмата извежда вътрешното представяне с дължина 32 или 64 бита на цяла число (без водещите нули), въведено от клавиатурата.

```
package Arithmetic;

import java.util.Scanner;

public class Integers {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int number = sc.nextInt();
        System.out.println("Bits: " + Integer.toBinaryString(number));
    }
}
```

Изпълнение на програмата:

Първо изпълнение:

```
Enter an integer: 32
Bits: 100000
```

Второ изпълнение:

```
Enter an integer: -32
Bits: 111111111111111111111111000000
```

Трето изпълнение:

```
Enter an integer: -1
Bits: 11111111111111111111111111111111
```

Програма IntegerTypeParameters: Програмата извежда параметрите – размерът на полето за представяне на стойностите (в битове), минималната и максималната стойности, за всеки от целочислените типове.

```
package Arithmetic;

public class IntegerTypeParameters {
```

```

public static void main(String[] args) {
    System.out.println("      byte");
    System.out.println("Byte.SIZE = " + Byte.SIZE);
    System.out.println("Byte.MIN_VALUE = " + Byte.MIN_VALUE);
    System.out.println("Byte.MAX_VALUE = " + Byte.MAX_VALUE);

    System.out.println("      short");
    System.out.println("Short.SIZE = " + Short.SIZE);
    System.out.println("Short.MIN_VALUE = " + Short.MIN_VALUE);
    System.out.println("Short.MAX_VALUE = " + Short.MAX_VALUE);

    System.out.println("      integer");
    System.out.println("Integer.SIZE = " + Integer.SIZE);
    System.out.println("Integer.MIN_VALUE = " + Integer.MIN_VALUE);
    System.out.println("Integer.MAX_VALUE = " + Integer.MAX_VALUE);

    System.out.println("      long");
    System.out.println("Long.SIZE = " + Long.SIZE);
    System.out.println("Long.MIN_VALUE = " + Long.MIN_VALUE);
    System.out.println("Long.MAX_VALUE = " + Long.MAX_VALUE);
}
}

```

Изпълнение на програмата:

```

byte
Byte.SIZE = 8
Byte.MIN_VALUE = -128
Byte.MAX_VALUE = 127
short
Short.SIZE = 16
Short.MIN_VALUE = -32768
Short.MAX_VALUE = 32767
integer
Integer.SIZE = 32
Integer.MIN_VALUE = -2147483648
Integer.MAX_VALUE = 2147483647
long
Long.SIZE = 64
Long.MIN_VALUE = -9223372036854775808
Long.MAX_VALUE = 9223372036854775807

```

2. Типове с плаваща точка float и double

Стойностите на типовете **float** и **double** се представят като двоични числа с плаваща точка. За представяне на такива числа се използват специфицираните в [IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985] 32-битов формат с единична точност и 64-битов формат с двойна точност за представяне на двоични числа с плаваща точка и операции с тях. Множеството от стойности на всеки от двета формата включва:

- Крайни положителни и отрицателни двоични числа с плаваща точка
- Положителна нула и отрицателна нула
- Положителна безкрайна стойност Infinity и отрицателна безкрайна стойност -Infinity
- Специални стойности NaN, които се използват за представяне на резултата на некоректна операция

С изключение на NaN, стойностите са наредени в реда: отрицателна безкрайна стойност -Infinity, отрицателни крайни ненулеви стойности, отрицателна и положителна нула, положителни крайни ненулеви стойности и положителна безкрайна стойност Infinity. Положителната нула и отрицателната нула се считат равни, но останалите операции ги различават.

Всяко крайно ненулево двоично число се представя като число с плаваща точка от вида $s.m.2^{(p-N+1)}$, където s е +1 или -1, мантисата m е положително цяло число по-малко от 2^N , порядъкът p е цяло число между $E_{min}=-(2^{K-1}-2)$ и $E_{max}=2^{K-1}-1$, включително, а N и K са параметри. Числото е представено в *нормализиран вид*, ако $m \geq 2^{(N-1)}$ и в *ненормализиран вид* в противен случай. В Таблица 5 са дадени стойностите на N , K , E_{max} и E_{min} за множеството от стойности с единична точност S_{float} и за множеството от стойности с двойна точност S_{double} , както и за още две множества $S_{float-extended-exponent}$ и $S_{double-extended-exponent}$.

Параметър	S_{float}	$S_{float-extended-exponent}$	S_{double}	$S_{double-extended-exponent}$
N	24	24	53	53
K	8	≥ 11	11	≥ 15
E_{max}	+127	$\geq +1023$	+1023	$\geq +16383$
E_{min}	-126	≤ -1022	-1022	≤ -16382

Таблица 5. Параметри на типовете с плаваща точка

Всяка реализация на езика Java, поддържа задължително стандартните множества S_{float} и S_{double} и поне едно от останалите две множества

Полето от паметта, в което се съхранява число с плаваща точка, се състои от две части – за мантисата и за порядъка. Размерът на полето за мантисата определя точността на числата, а размерът на полето за порядъка – големината. Елементите на множеството S_{float} се използват за представяне на стойностите на типа **float**, а елементите на множеството S_{double} – за представяне на стойностите на типа **double**, като:

- За типа **float** полето за мантисата се състои от 23 бита и точността е 23 двоични цифри (6-7 десетични цифри), а порядъкът има обхват от 2^{-126} до 2^{+127} (приблизително от 10^{-45} до 10^{+38})
- За типа **double** полето за мантисата се състои от 52 бита и точността е 52 двоични цифри (13-14 десетични цифри), а порядъкът има обхват от 2^{-1022} до 2^{+1023} (приблизително от 10^{-324} до 10^{+308})

Елементите на множествата $S_{float-extended-exponent}$ и $S_{double-extended-exponent}$ се използват единствено за представяне на междинните резултати при изчисляване на изразите от тип **float** или **double**, като окончателната стойност се преобразува в стойност на типа **float** или **double**, съгласно т.6.1.5.

Операциите, които могат да се прилагат над тези стойности, са:

- Операции, чийто резултат е стойност на типа **float** или **double**
 - Операции за смяна на знака + и -
 - Операции за умножение *, деление / и деление по модул %
 - Операции за събиране + и изваждане -
 - Операция за увеличаване с 1 ++ (префиксна и постфиксна форма)
 - Операция за намаляване с 1 -- (префиксна и постфиксна форма)
 - Операции за присвояване =, *=, /=, %=, +=, -=
- Операции за сравнение ==, !=, <, <=, > и >=, чийто резултат е стойност на типа **boolean**
- Условна операция ?:
- Операция за преобразуване на число с плаваща точка в стойност на указан числов тип
- Операция за конкатениране на низове +

При изпълнението на операциите +, - и * се изчисляват съответно сумата, разликата и произведението на двета операнда от плаващ тип, а при изпълнението на операциите / и % се изчисляват съответно частното и остатъкът при делението на левия операнд на десния. Получената стойност, съгласно [IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985], се „закръглява” до най-близката стойност на типа на резултата и закръглената стойност е резултатът на операцията. Таблица 6 съдържа

результатите от изпълнението на аритметичните операции за различни стойности на операндите A и B от тип **float**.

A	B	A+B	A-B	A*B	(double) A/B	(float) A/B	A%B
7.0	3.0	10.0	4.0	21.0	2.3333333333333335	2.3333333	1.0
7.0	-3.0	4.0	10.0	-21.0	-2.3333333333333335	-2.3333333	1.0
-7.0	3.0	-4.0	-10.0	-21.0	-2.3333333333333335	-2.3333333	-1.0
-7.0	-3.0	-10.0	-4.0	21.0	2.3333333333333335	2.3333333	-1.0
1	0	1.0	1.0	0.0	Infinity	Infinity	NaN

Таблица 6. Резултати на аритметични операции за типовете с плаваща точка

Резултатите от изпълнението на операциите за смяна на знака за различни стойности на операнда A от тип **float** са дадени в Таблица 7.

A	+A	-A
5.0	5.0	-5.0
-5.0	-5.0	5.0
NaN	NaN	NaN
Infinity	Infinity	-Infinity
0.0	0.0	-0.0
-0.0	-0.0	0.0

Таблица 7. Резултати на операции за смяна на знака за типовете с плаваща точка

Резултатите от изпълнението на операциите за присвояване, увеличаване и намаляване с единица за A=7.0 и B=3.0 от тип **float** и новата стойност на A са дадени в Таблица 8.

	A+=B	A-=B	A*=B	A/=B	A%=B	A=B	A++	++A	A--	--A
Резултат	10.0	4.0	21.0	2.3333333	1.0	3.0	7.0	8.0	7.0	7.0
A	10.0	4.0	21.0	2.3333333	1.0	3.0	8.0	8.0	6.0	7.0

Таблица 8. Резултати на операции за присвояване, увеличаване и намаляване с единица за типове с плаваща точка

Класовете **Float**, **Double** и **Math** от пакета `java.lang`, предлагат още възможности за обработка на числа с плаваща точка чрез декларирани в тях конструктори, методи и константи - виж <http://java.sun.com/javase/6/docs/api/>.

Програма Floats: Програмата извежда вътрешното представяне с дължина 32 или 64 бита (без водещите нули):

- На NaN
- На -Infinity
- На +Infinity
- На число с плаваща точка, въведено от клавиатурата

```
package Arithmetic;

import java.util.Scanner;

public class Floats {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        float number = Float.NaN;
        int bits = Float.floatToIntBits(number);
        System.out.println("NaN: " + Integer.toBinaryString(bits));

        number = Float.NEGATIVE_INFINITY;
        bits = Float.floatToIntBits(number);
        System.out
```

```
        .println("NEGATIVE_INFINITY: " + Integer.toBinaryString(bits));

    number = Float.POSITIVE_INFINITY;
    bits = Float.floatToIntBits(number);
    System.out
        .println("POSITIVE_INFINITY: " + Integer.toBinaryString(bits));

    System.out.print("      Enter a float: ");
    number = sc.nextFloat();
    System.out.println(number);
    bits = Float.floatToIntBits(number);
    System.out.println("Bits: " + Integer.toBinaryString(bits));
}
```

Изпълнение на програмата:

Първо изпълнение:

```
NaN: 11111111000000000000000000000000  
NEGATIVE_INFINITY: 11111111000000000000000000000000  
POSITIVE_INFINITY: 11111111000000000000000000000000  
    Enter a float: 0,085  
0.085  
Bits: 1111011010110000101000111011
```

Второ изпълнение:

```
NaN: 11111111000000000000000000000000  
NEGATIVE_INFINITY: 11111111000000000000000000000000  
POSITIVE_INFINITY: 11111111000000000000000000000000  
Enter a float: -118.625  
-118.625  
Bits: 11000010111011010100000000000000
```

Програма RealTypeParameters: Програмата извежда параметрите – размерът на полето за представяне на стойностите (в битове), минималната и максималната стойности и други, за всеки от типовете с плаваща точка.

```
package Arithmetic;

public class RealTypeParameters {
    public static void main(String[] args) {
        System.out.println("      float");
        System.out.println("Float.SIZE = " + Float.SIZE);
        System.out.println("Float.MAX_EXPONENT = " + Float.MAX_EXPONENT);
        System.out.println("Float.MAX_VALUE = " + Float.MAX_VALUE);
        System.out.println("Float.MIN_EXPONENT = " + Float.MIN_EXPONENT);
        System.out.println("Float.MIN_NORMAL = " + Float.MIN_NORMAL);
        System.out.println("Float.MIN_VALUE = " + Float.MIN_VALUE);
        System.out.println("Float.NaN = " + Float.NaN);
        System.out.println("Float.NEGATIVE_INFINITY = "
            + Float.NEGATIVE_INFINITY);
        System.out.println("Float.POSITIVE_INFINITY = "
            + Float.POSITIVE_INFINITY);

        System.out.println("      double");
        System.out.println("Double.SIZE = " + Double.SIZE);
        System.out.println("Double.MAX_EXPONENT = " + Double.MAX_EXPONENT);
        System.out.println("Double.MAX_VALUE = " + Double.MAX_VALUE);
        System.out.println("Double.MIN_EXPONENT = " + Double.MIN_EXPONENT);
        System.out.println("Double.MIN_NORMAL = " + Double.MIN_NORMAL);
        System.out.println("Double.MIN_VALUE = " + Double.MIN_VALUE);
        System.out.println("Double.NaN = " + Double.NaN);
```

```

        System.out.println("Double.NEGATIVE_INFINITY = "
            + Double.NEGATIVE_INFINITY);
        System.out.println("Double.POSITIVE_INFINITY = "
            + Double.POSITIVE_INFINITY);
    }
}

```

Изпълнение на програмата:

```

float
Float.SIZE = 32
Float.MAX_EXPONENT = 127
Float.MAX_VALUE = 3.4028235E38
Float.MIN_EXPONENT = -126
Float.MIN_NORMAL = 1.17549435E-38
Float.MIN_VALUE = 1.4E-45
Float.NaN = NaN
Float.NEGATIVE_INFINITY = -Infinity
Float.POSITIVE_INFINITY = Infinity
double
Double.SIZE = 64
Double.MAX_EXPONENT = 1023
Double.MAX_VALUE = 1.7976931348623157E308
Double.MIN_EXPONENT = -1022
Double.MIN_NORMAL = 2.2250738585072014E-308
Double.MIN_VALUE = 4.9E-324
Double.NaN = NaN
Double.NEGATIVE_INFINITY = -Infinity
Double.POSITIVE_INFINITY = Infinity

```

Програма MathFunctions: Програмата извежда стойностите на константите на класа **Math**, както и резултатите от изпълнението на някои от методите на класа:

```

package Arithmetic;

public class MathFunctions {
    public static void main(String[] args) {
        System.out.println("      Math constants");
        System.out.println("E = " + Math.E);
        System.out.println("PI = " + Math.PI);

        System.out.println("      Math functions");
        double x = Math.random() * 100;
        System.out.println("Random number created: " + x);
        System.out.println("ceil(" + x + ") = " + Math.ceil(x));
        System.out.println("floor(" + x + ") = " + Math.floor(x));
        System.out.println("round(" + x + ") = " + Math.round(x));
        System.out.println("square root of " + x + " = " + Math.sqrt(x));
        System.out.println("pow(" + Math.sqrt(x) + ",2) = "
            + Math.pow(Math.sqrt(x), 2));
        System.out.println("cube root of " + x + " = " + Math.cbrt(x));
        System.out.println("pow(" + Math.cbrt(x) + ",3) = "
            + Math.pow(Math.cbrt(x), 3));
        System.out.println("log(" + x + ") = " + Math.log(x));
        System.out.println("exp(" + Math.log(x) + ") = "
            + Math.exp(Math.log(x)));

        x = Math.PI / 3.0;
        System.out.println("cos(" + x + ") = " + Math.cos(x));
        System.out.println("acos(" + Math.cos(x) + ") = "
            + Math.acos(Math.cos(x)));
        System.out.println("sin(" + x + ") = " + Math.sin(x));
        System.out.println("asin(" + Math.sin(x) + ") = "
            + Math.asin(Math.sin(x)));
        System.out.println("tan(" + x + ") = " + Math.tan(x));
    }
}

```

```
System.out.println("atan(" + Math.tan(x) + ") = "
+ Math.atan(Math.tan(x)));
}
```

Изпълнение на програмата:

```
Math constants
E = 2.718281828459045
PI = 3.141592653589793
Math functions
Random number created: 14.263150933618496
ceil(14.263150933618496) = 15.0
floor(14.263150933618496) = 14.0
round(14.263150933618496) = 14
square root of 14.263150933618496 = 3.7766586996468843
pow(3.7766586996468843,2) = 14.263150933618494
cube root of 14.263150933618496 = 2.4251493688342243
pow(2.4251493688342243,3) = 14.263150933618492
log(14.263150933618496) = 2.657679353664254
exp(2.657679353664254) = 14.263150933618498
cos(1.0471975511965976) = 0.5000000000000001
acos(0.5000000000000001) = 1.0471975511965976
sin(1.0471975511965976) = 0.8660254037844386
asin(0.8660254037844386) = 1.0471975511965976
tan(1.0471975511965976) = 1.7320508075688767
atan(1.7320508075688767) = 1.0471975511965976
```

3. Числови константи

3.1. Целочислени константи

Целочислените константи са три вида:

- **Десетична** константа е 0 или последователност от една или повече цифри, принадлежащи на множеството {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, незапочваща с 0 и представя десетично число по-голямо или равно на нула
- **Шестнадесетична** константа започва с 0x или 0X, след които има поне един символ от множеството {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, A, B, C, D, E, F} и представя цяло десетично число
- **Осмична** константа започва с 0, след която има поне един символ от множеството {0, 1, 2, 3, 4, 5, 6, 7} и представя цяло десетично число

Целочислена константа е от тип **long**, ако завършва с буквата L или l. В противен случай тя е от тип **int**.

Най-голямата десетична константа от тип **int** е 2147483648, а за тип **long** е 9223372036854775808L и те могат да бъдат операнди само на операцията унарен -.

Най-големите положителни шестнадесетична и осмична константи от тип **int** са 0x7fffffff и 017777777777 съответно, и всяка представя десетичното число 2147483647 ($2^{31}-1$). Най-малките отрицателни шестнадесетична и осмична константи от тип **int** са 0x80000000 и 020000000000 съответно и всяка представя десетичното число -2147483648 (- 2^{31}). Шестнадесетичната константа 0xffffffff и осмичната константа 037777777777 представят числото -1.

По време на транслация възниква грешка, ако десетична константа от тип **int** по-голяма от 2147483648 или 2147483648 се използва не само като операнд на операцията унарен -, както и ако шестнадесетична или осмична константа не може да се представи в 32 бита.

3.2. Константи с плаваща точка

Константите с плаваща точка са реални десетични числа, състоящи се от следните части: цяла част, точка, дробна част, експонента и суфикс, определящ нейния тип. Експонентата се представя чрез символа e или E, след който следва цяло число със или без знак.

Най-голямата положителна константа от тип **float** е 340282347e+38f, а най-малката положителна ненулева константа е 1.40239846e-45f. Най-голямата положителна константа от тип **double** е 1.79769313486231570e+308, а най-малката положителна ненулева константа е 4.94065645841246544e-324.

По време на транслация възниква грешка, ако константата е твърде голяма, при което нейното вътрешно представяне става **Infinity**. В програмата може да се използват такива стойности, без да възникне грешка при транслация, чрез използване на изрази като 1f/0f или -1d/0d, или чрез използване на предварително дефинираните константи **POSITIVE_INFINITY** и **NEGATIVE_INFINITY** на класовете **Float** и **Double**.

По време на транслация възниква грешка, ако ненулева константа е твърде малка, при което нейното вътрешно представяне е това на числото нула. Не възниква грешка в случай, че вътрешното й представяне е ненормализирано число.

4. Локални променливи от числов тип

Декларацията на локална променлива от числов тип има вида *Тип Име_на_променлива;* или *Тип Име_на_променлива = Инициализатор;*

Типът е **byte**, **short**, **int**, **long**, **float** или **double**, а инициализаторът – *аритметичен израз*. При наличие на инициализатор на променливата се присвоява стойността на аритметичния израз. Начална стойност на променлива може да се присвои и чрез *оператор за присвояване* от вида *Числова_променлива = Аритметичен_израз;*. Текущата стойност на променлива може да се промени чрез операторите:

Оператор_за_присвояване
++ Числова_променлива;
Числова_променлива ++;
-- Числова_променлива;
Числова_променлива --;

Оператор_за_присвояване:
Числова_променлива Операция за присвояване Аритметичен_израз;

Всеки от тях е от вида *Аритметичен_израз*; и при изпълнението му се пресмята стойността на аритметичния израз, в резултат на което се променя стойността на променливата.

5. Операция за преобразуване на типа, аритметични операции, операции за присвояване и аритметични изрази

Операцията за преобразуване на типа има вида *(Числов тип)A*. Ако операндът A не е от числов тип, се получава грешка по време на транслация. По време на изпълнение стойността

на операнда се превръща в стойност на посочения в скобите тип, съгласно т.6.2.3. Получената стойност е резултатът на операцията.

Операциите за смяна на знака имат вида $+A$ и $-A$. По време на транслация се получава грешка, ако операндът A на операция не е от числов тип. По време на изпълнение операндът се преобразува, съгласно т.6.2.5. Определеният тип е типът на резултата. Преобразуваната стойност на операнда е резултатът на операцията +. Ако преобразуваната стойност, с противоположен знак, е допустима за типа на резултата, тя е резултатът на операцията -. В противен случай, тя се превръща в стойност на типа на резултата, съгласно т.6.1.3 и получената стойност е резултатът на операцията.

Всяка от операциите +, -, *, / и % има вида $A\text{Operаци}яB$. Ако единият от операндите A и B на операцията + е от тип **String**, се изпълнява операцията за конкатениране на низове като, ако другият операнд е от числов тип, той се превръща в стойност на типа **String**, съгласно т.6.2.4. Резултатът на операцията е конкатенацията на двата низа. В противен случай това е аритметичната операция +. По време на транслация се получава грешка, ако двата операнда на аритметична операция не са от числов тип. По време на изпълнение двата операнда се превръщат в един и същ тип, съгласно т.6.2.5. Ако двата операнда са от целочислен тип, се изпълнява съответната операция на целочисления тип. В противен случай се изпълнява съответната операция на типа с плаваща точка.

Операциите за присвояване имат два операнда. Те трябва да са от числов тип, като левият операнд трябва да е променлива. В противен случай, се получава грешка по време на транслация. Типът на резултата на операция за присвояване е типът на променливата. По време на изпълнение на операцията за присвояване $A=B$:

1. Стойността на операнда B се превръща в стойност на типа на променливата A, съгласно т.6.2.1. Ако е необходимо да се извърши стесняващо преобразуване, то типът на променливата трябва да се укаже явно, т.е. $A=(\text{Tip_на_}A)B$. В противен случай, се получава грешка по време на транслация
2. Преобразуваната стойност се присвоява на променливата. Тази стойност е резултатът на операцията

Изпълнението на операция за присвояване $A\omega B$, където $\omega \in \{+=, -=, *=, /=, \%=\}$ е еквивалентно на изпълнението на $A=(\text{Tip_на_}A)(A\omega B)$.

Операциите ++ и – имат един операнд A, който трябва да е променлива от числов тип. В противен случай, се получава грешка по време на транслация. Типът на резултата е типът на операнда. По време на изпълнение операндът се използва за определяне на резултата и се променя, като промяната се състои в изпълнението на $A=(A+1)$ и $A=(A-1)$, съответно. При префиксните форми $++A$ и $-A$ първо се променя A и новата стойност е резултатът на операцията, а при постфиксните форми $A++$ и $A-$ резултатът е стойността на A и след това A се променя.

Аритметичните изрази се съставят и пресмятат, съгласно спецификацията на езика Java. Като резултат от пресмятането на аритметичен израз се получава стойност. Всеки аритметичен израз има тип, който се определя по време на транслация и резултатът от изчисленията е стойност на този тип.

Редът на изпълнение на операциите в аритметичен израз може да се зададе явно чрез използване на скоби. Ако няма такива, операциите се изпълняват съгласно техния приоритет, като:

- Операция с по-висок приоритет се изпълнява преди операция с по-нисък приоритет
- Операции с един и същ приоритет се изпълняват, съгласно тяхното асоциативно правило

Таблица 9 съдържа всички операции в Java, наредени в намаляващ ред на приоритетите им и асоциативното правило за всяка група операции с един и същ приоритет.

Постфиксни операции	[] . () ++ --	Отляво надясно
Унарни (префиксни) операции	++ -- + - ~ !	Отляво наляво
Създаване и преобразуване на типа	new (тип)израз	Отляво наляво
Мултиликативни операции	* / %	Отляво надясно
Адитивни операции	+ -	Отляво надясно
Операции за изместване	<< >> >>>	Отляво надясно
Отношения	< > <= >= instanceof	Отляво надясно
Отношения	== !=	Отляво надясно
Побитова конюнкция	&	Отляво надясно
Побитова сума по модул две	^	Отляво надясно
Побитова дизюнкция		Отляво надясно
Конюнкция	&&	Отляво надясно
Дизюнкция		Отляво надясно
Условна операция	? :	Отляво наляво
Операции за присвояване	= += -= *= /= %= &= ^= = <=>= >>=	Отляво наляво

Таблица 9. Приоритет на операциите

Операндите на аритметична операция се изчисляват преди изпълнението на самата операция, като операндите на бинарна операция се изчисляват *отляво надясно*.

Аритметичен израз, чиято стойност се определя по време на транслация, се нарича *константен аритметичен израз*. В него може да участват само:

- Числови константи
- Операции за преобразуване в числови типове
- Операции за смяна на знака
- Мултиликативни операции *, / и %
- Адитивни операции + и –
- Именувани константи, чиито инициализатори са константни изрази

Следващите програми, изясняват особеностите на целочислената аритметика и на аритметиката с плаваща точка.

Програма Example1_5:

```
package Arithmetic;

public class Example1_5 {
    public static void main(String[] args) {
        byte a = 0177;
        System.out.println("a = " + a);
        byte b = (byte) (a + 1);
        System.out.println("a + 1 = " + b);

        a = -128;
        System.out.println("a = " + a);
        b = (byte) (a - 1);
        System.out.println("a - 1 = " + b);
    }
}
```

Изпълнение на програмата:

```
a = 127
a + 1 = -128
a = -128
```

```
a - 1 = 127
```

Осмичната константа 0177 е от тип **int** и представя десетичното число 127. По време на транслация тя се преобразува в тип **byte** и се присвоява на променливата a. Резултатът a+1=128 е от тип **int** и се представя в поле от 32 бита чрез двоичната конфигурация 0...010000000. Той се преобразува в тип **byte**, като младшите 8 бита 10000000 се записват в полето на променливата b. Съответното десетично число е отрицателно, тъй като първият бит е 1 и неговата абсолютна стойност се получава по следния начин: от 10000000 се вади 1; полученият резултат 01111111 се инвертира и инвертираното число 10000000 е двоичен запис на 128, т.e. b=-128.

По време на транслация се изчислява изразът -128 , като резултатът е стойност на типа **int**, защото десетичната константа 128 е от тип **int**. Получената стойност се преобразува в стойност на типа **byte** и се присвоява на променливата a. Резултатът $a-1=-129$ е от тип **int** и се представя в поле от 32 бита чрез двоичната конфигурация 1...101111111. Тя се получава по следния начин: двоичното представяне на 129 в 32 бита 0...010000001 се инвертира и към полученото 1...10111110 се добавя 1. Полученото 1...10111111 се преобразува в тип **byte**, като се вземат младшите 8 бита 01111111 и се записват в полето на променливата b. Съответното десетично число е положително, тъй като първият бит е 0 и това е неговия двоичен запис, т.e. $b=127$.

Програма Example1_6:

```
package Arithmetic;

public class Example1_6 {
    public static void main(String[] args) {
        System.out.println("      Overflow");
        System.out.println("1e308 * 10 = " + 1e308 * 10);

        System.out.println("      Gradual underflow");
        double a = Math.PI * 1e-305;
        System.out.println("Math.PI * 1e-305 = " + a);
        System.out.println("Math.PI * 1e-310 = " + (a /= 1e5));
        System.out.println("Math.PI * 1e-315 = " + (a /= 1e5));
        System.out.println("Math.PI * 1e-320 = " + (a /= 1e5));
        System.out.println("Math.PI * 1e-325 = " + (a /= 1e5));

        System.out.println("      Double to int");
        System.out.println("(int)12345.6 = " + (int) 12345.6);
        System.out.println("(int)(-12345.6) = " + (int) (-12345.6));
        System.out.println("Math.round(12345.6) = " + Math.round(12345.6));
        System.out.println("Math.round(-12345.6) = " + Math.round(-12345.6));
    }
}
```

Изпълнение на програмата:

```
Overflow
1e308 * 10 = Infinity
Gradual underflow
Math.PI * 1e-305 = 3.141592653589793E-305
Math.PI * 1e-310 = 3.1415926535898E-310
Math.PI * 1e-315 = 3.141592653E-315
Math.PI * 1e-320 = 3.142E-320
Math.PI * 1e-325 = 0.0
      Double to int
(int)12345.6 = 12345
(int)(-12345.6) = -12345
Math.round(12345.6) = 12346
Math.round(-12345.6) = -12346
```

От резултатите, които се получават при изпълнението на програмата се вижда, че при много големи и малки стойности, както и при преобразуване в целочислен тип, може да се получи загуба на точност.

6. Преобразуване на типовете

Всеки израз има тип, който се определя от неговата структура и от типовете на неговите операнди по време на транслация. Това позволява по време на транслация всеки израз от тип S да се разглежда като израз от тип T. Типът T може да се укаже явно от програмиста или да се определи автоматично, съгласно възприети правила. По време на изпълнение стойността на израза се превръща в стойност на типа T, чрез прилагане на някое от следните специфични преобразувания, като в някои случаи първо се проверява дали това е допустимо.

- Тъждествени преобразувания
- Разширяващи преобразувания между примитивни типове
- Стесняващи преобразувания между примитивни типове
- Разширяващи преобразувания между съставни типове
- Стесняващи преобразувания между съставни типове
- Преобразувания в тип String
- Преобразувания между различни множества от стойности

6.1. Специфични преобразувания между примитивни типове

6.1.1. Тъждествени преобразувания

Преобразуването от тип T в същия тип е разрешено за всеки тип T.

6.1.2. Разширяващи преобразувания

Разрешени са следните преобразувания:

- От byte в short, int, long, float или double
- От short в int, long, float или double
- От char в int, long, float или double
- От int в long, float или double
- От long във float или double
- От float в double

Стойност на числов тип S, за който се отделят N бита, се преобразува в стойност на числов тип T, за който се отделят M бита и $N < M$, като:

- Ако S и T са целочислени типове, то се състои в разширяване със знаковия бит на N-битовото представяне на стойността до M-битово представяне, при което стойността не се променя
 - Ако S е целочислен, а T е тип с плаваща точка, то стойността се закръглява до най-близката стойност на T. Това може да доведе до загуба на точност при превръщане от тип **int** или **long** в тип **float** или от тип **long** в тип **double**
 - Ако S и T са типове с плаваща точка, стойността се запазва, с изключение на случаите, когато това е стойност на константен израз

По време на изпълнение тези преобразувания не активират изключения, въпреки възможната загуба на точност.

Програма Example1_7: Програмата изпълнява разширяващи преобразувания

```

package Arithmetic;

public class Example1_7 {
    public static void main(String[] args) {
        int big = 1234567890;
        float approx = big;
        System.out.println(big - (int) approx);
    }
}

```

Изпълнение на програмата:

-46

Програмата превръща стойността 1234567890 на типа **int** в стойност на типа **float**. Получената стойност се преобразува обратно в стойност на типа **int** и се извежда от 1234567890. Програмата извежда -46, вместо 0, защото полето за мантиса не може да побере 10-те цифри на цялото число.

6.1.3. Стесняващи преобразувания

Разрешени са преобразуванията:

- От byte в char
- От short в byte или char
- От char в byte или short
- От int в byte, short или char
- От long в byte, short, char или int
- От float в byte, short, char, int или long
- От double в byte, short, char, int, long или float

Стойност на числов тип S, за който се отделят N бита, се превръща в стойност на числов тип T, за който се отделят M бита и $N > M$, като:

- Ако S и T са целочислени типове, то преобразуваната стойност се представя чрез младшите M бита на N-битовото представяне, в резултат на което може да се получи загуба на информация за големината и знака на стойността.
- Ако S е тип с плаваща точка, а T е целочислен тип, то:
 1. Стойността се превръща в тип **long**, ако T е **long**, в противен случай - в тип **int**, като се „отрязва” дробната й част
 2. Ако T е **int** или **long**, то стойността, получена на стъпка 1, е резултатът от преобразуването. В противен случай стойността, получена на стъпка 1, се превръща в стойност на типа T чрез стесняващо преобразуване и това е резултатът от преобразуването

Въпреки възможната загуба на информация за стойността, тези преобразувания не активират изключения по време на изпълнение.

Програма Example1_8: Програмата изпълнява стесняващи преобразувания на отрицателната и положителната безкрайни стойности на типа **float** в стойности на всеки от целочислените типове.

```

package Arithmetic;

public class Example1_8 {
    public static void main(String[] args) {
        float fmin = Float.NEGATIVE_INFINITY;
        float fmax = Float.POSITIVE_INFINITY;
        System.out.println("long: " + (long) fmin + "..." + (long) fmax);
        System.out.println("int: " + (int) fmin + "..." + (int) fmax);
    }
}

```

```
System.out.println("short: " + (short) fmin + ".." + (short) fmax);
System.out.println("char: " + (int) (char) fmin + ".."
    + (int) (char) fmax);
System.out.println("byte: " + (byte) fmin + ".." + (byte) fmax);
}
}
```

Изпълнение на програмата:

```
long: -9223372036854775808..9223372036854775807
int: -2147483648..2147483647
short: 0..-1
char: 0..65535
byte: 0..-1
```

От получените резултати се вижда, че за типовете **char**, **int** и **long** преобразуваните стойности представляват най-малката и най-голямата стойности на типа. За типовете **byte** и **short** преобразуването се осъществява на две стъпки, като на втората стъпка се изпълнява стесняващо преобразуване от тип **int** в тип **byte** или **short**, което води до промяна на стойността.

6.1.4. Преобразования в тип **String**

Преобразуването от тип T в тип **String** е разрешено за всеки тип T.

6.1.5. Преобразуване между различни множества от стойности с плаваща точка

Стойност на аритметичен константен израз, се преобразува по следния начин:

- Ако стойността е от тип **float** и не принадлежи на S_{float} , тя се представя чрез най-близката стойност на множеството S_{float}
- Ако стойността е от тип **double** и не принадлежи на S_{double} , тя се представя чрез най-близката стойност на множеството S_{double}

Всяка реализация на езика Java поддържа по избор някое от следните преобразувания на стойност на израз, които се прилагат само ако са допустими за израза:

- Ако стойността е елемент на множеството $S_{\text{float-extended-exponent}}$, тя се представя чрез най-близката стойност на множеството S_{float}
- Ако стойността е елемент на множеството $S_{\text{double-extended-exponent}}$, тя се представя чрез най-близката стойност на множеството S_{double}

И за двата вида изрази, ако стойността е много голяма, това може да доведе до представянето й чрез безкрайна стойност **Infinity** със същия знак, а ако е много малка – до представянето й с ненормализирано число или нула със същия знак, при което има загуба на точност.

6.1.6. Забранени преобразувания

Не са разрешени преобразувания:

- От съставен тип в примитивен тип
- От примитивен тип в съставен тип, освен в тип **String**

6.2. Автоматични преобразувания между примитивни типове

6.2.1. Преобразуване при присвояване

Това преобразуване се изпълнява, когато стойността на израз трябва да се присвои на променлива. То се състои в превръщането на стойността на израза в стойност на типа на

променливата, ако това е допустимо, като са разрешени само тъждествените или разширяващите преобразувания. Стесняващите преобразувания са разрешени само за целочислени константни изрази, като техният тип не е **long** и типът на променливата не е **int** или **long**. Ако преобразуването не е допустимо, по време на транслация се получава грешка.

Ако типът на променливата е **float** или **double**, то за получената стойност се прилага преобразуването от т. 6.1.5. Не се активират изключения по време на изпълнение.

Програма Example1_10: Програмата съдържа примери на преобразувания, които се извършват при присвояване.

```
package Arithmetic;

public class Example1_10 {
    public static void main(String[] args) {
        short s = 12; // narrow 12 to short
        float f = s; // widen short to float
        System.out.println("f=" + f);

        f = 1.23f;
        double d = f; // widen float to double
        System.out.println("d=" + d);
    }
}
```

Изпълнение на програмата:

```
f=12.0
d=1.2300000190734863
```

6.2.2. Преобразуване при извикване на метод

Това преобразуване се прилага за всеки фактически параметър и съответния му формален параметър при извикване на метод или конструктор, като се изпълняват същите действия, както при преобразуване при присвояване. Не са разрешени нейни стесняващи преобразувания.

6.2.3. Явно преобразуване

Това преобразуване се прилага само върху операнда на операцията за преобразуване на типа. По време на изпълнение стойността на операнда се превръща в стойност на указания тип, като са разрешени тъждествените преобразувания и преобразуванията между примитивни типове. След това за получената стойност се прилага преобразуването от т. 6.1.5. В някои случаи по време на транслация се получава грешка.

6.2.4. Преобразуване в String

Стойност x на примитивен тип T се преобразува първо в стойност на съставен тип, като:

- Ако T е **boolean**, се изпълнява new Boolean(x) на класа **Boolean**
- Ако T е **char**, се изпълнява new Character(x) на класа **Character**
- Ако T е **byte**, **short** или **int**, се изпълнява new Integer(x) на класа **Integer**
- Ако T е **long**, се изпълнява new Long(x) на класа **Long**
- Ако T е **float**, се изпълнява new Float(x) на класа **Float**
- Ако T е **double**, се изпълнява new Double(x) на класа **Double**

След това получената стойност се преобразува в стойност на типа **String** чрез извикване на метода **toString**, съответно на класа **Boolean**, **Character**, **Integer**, **Long**, **Float** и **Double**.

6.2.5. Преобразуване на operandите на числова операция

Това преобразуване се прилага над operandите на аритметична операция, като са разрешени тъждествените или разширяващите преобразувания между примитивни типове. То се използва за превръщане на operandите в един и същи тип, така че да може да се изпълни самата операция.

Ако операцията има един operand и неговият тип по време на транслация е **byte**, **short** или **char**, operandът се превръща в тип **int** чрез разширяващо преобразуване. В противен случай, остава същия. В някои случаи за получената стойност се прилага преобразуването от т. 6.1.5.

Ако операцията има два операнда, то:

1. Ако единият от operandите е от тип **double**, то другият се преобразува в тип **double**
2. В противен случай, ако единият от operandите е от тип **float**, то другият се преобразува в тип **float**
3. В противен случай, ако единият от operandите е от тип **long**, то другият се преобразува в тип **long**
4. В противен случай, двата operandи се преобразуват в тип **int**

В някои случаи за operandите се прилага преобразуването от т. 6.1.5.



Задани за упражнение

Задача: Напишете програма **Test1**, която чете две цели числа A и B, въведени от клавиатурата и извежда тяхната сума, разлика, произведение, частно и остатък при делението им. Изпълнете програмата с посочените входни данни и попълнете получените резултати в Таблица 1. Коментирайте получените резултати.

A	B	A+B	A-B	A*B	A/B	A%B
5	2					
5	-2					
-5	2					
-5	-2					
-2147483648	-1					
0	23					
23	0					
3	5					
2147483647	1					

Таблица 1. Резултати на аритметични целочислени операции

Задача: Напишете програма **Test2**, която чете две реални числа A и B, въведени от клавиатурата и извежда тяхната сума, разлика, произведение, частно и остатък при делението им. Изпълнете програмата с посочените входни данни и попълнете получените резултати в Таблица 2. Коментирайте получените резултати.

A	B	A+B	A-B	A*B	(float) (A/B)	(double) (A/B)	A%B
4.5	3.0						
0.45	0.03						
45.0	3.0						
1.0	0.0						
-1.0	0.0						
1.0	-0.0						
0.0	0.0						

1.0e308 10.0

Таблица 2. Резултати на аритметични операции с плаваща точка

Задача: Обяснете защо програмата **Test3** извежда такъв резултат. Може ли да се промени програмата, така че тя да извежда очакваното число?

```
package Arithmetic;

public class Test3 {
    public static void main(String[] args) {
        float f = 100000000;
        f = f + 1.0f;
        System.out.println("f = " + f);
    }
}
```

Изпълнение на програмата:

f = 1.0E8

Задача: Обяснете защо програмата **Test4** извежда такъв резултат.

```
package Arithmetic;

public class Test4 {
    public static void main(String[] args) {
        double f = 20000000000000.13;
        double g = 20000000000000.02;
        double h = f - g;
        System.out.println("h = " + h);
        int n = (int) (100 * h);
        System.out.println("n = " + n);
    }
}
```

Изпълнение на програмата:

h = 0.109375
n = 10

Задача: Обяснете защо програмата **Test5** извежда такъв резултат.

```
package Arithmetic;

public class Test5 {
    public static void main(String[] args) {
        double a = 100000000;
        double b = a * a + 1;
        double c = a * a;
        double d = b - c;
        System.out.println("d = " + d);
    }
}
```

Изпълнение на програмата:

d = 0.0

Задача: Обяснете защо програмата **Test6** извежда такъв резултат.

```
package Arithmetic;
```

```
public class Test6 {
    public static void main(String[] args) {
        int i = 1000000;
        System.out.println(i * i);
        long l = i;
        System.out.println(l * l);
        System.out.println(20296 / (l - i));
    }
}
```

Изпълнение на програмата:

```
Exception in thread "main" java.lang.ArithmetiсException: / by zero
at Arithmetic.Test6.main(Test6.java:9)
-727379968
1000000000000
```

Задача: Да се състави програма, която чете разстояние в метри и го превръща в мили, футове и инчове.

Задача: Да се състави програма, която чете дължина на радиус и пресмята:

- Лицето на кръга и дължината на окръжността с този радиус
- Обема на кълбото и повърхността на сферата с този радиус

Задача: Да се състави програма, която чете цяло четирицифрене число и:

- Отпечатва цифрите му с интервали между тях
- Определя сумата и произведението на цифрите на числото
- Определя неговото обратно число
- Определя числото, което се получава чрез размяна на първата и последната му цифри

Задача: От трицифрене число x извадили последната му цифра. Когато резделили резултата на 10 и към полученото частно добавили вляво последната цифра на числото x, получили числото 237. Да се състави програма, която определя числото x.

Задача: В трицифрене число x задраскали последната му цифра. Когато разменили местата на останалите две цифри и добавили вляво задрасканата цифра, получили числото n. Да се състави програма, която по зададено число n определя числото x.

Задача: Да се състави програма, която пресмята $f(x)$ за дадена стойност на променливата:

$$f(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}$$

Задача: Да се състави програма, която пресмята $f(x)$ за дадена стойност на променливата:

$$f(x) = x + \frac{1}{2} \cdot \frac{x^3}{3} + \frac{1 \cdot 3}{2 \cdot 4} \cdot \frac{x^5}{5} + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} \cdot \frac{x^7}{7}$$

Задача: Редицата от числа a_1, \dots, a_n, \dots се определя по формулите:

$a_1 = 6, a_2 = 11$
$a_n = 3a_{n-1} - 2a_{n-2}, n > 2$

Да се състави програма, която пресмята a_6 .

Задача: Да се състави програма, която за дадена аритметична прогресия $P(a_1, d, n) = a_1, \dots, a_n$

с първи член a_1 и разлика d

- Пресмята сумата на нейните членове
- Пресмята $a_{11} - 33a_2 + a_{21}$

Задача: Да се състави програма, която за даден полином на една променлива

$$P(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

- Пресмята стойността на полинома за дадена стойност на променливата
- Извежда полинома в символен вид:

$$P(x) = a5 * x ^ 5 + a4 * x ^ 4 + a3 * x ^ 3 + a2 * x ^ 2 + a1 * x + a0$$



Съставените до момента програми се изпълняват последователно, оператор след оператор, от началото на програмата до края. Твърде често обаче алгоритъмът, който се реализира, изиска:

- Проверка на дадено *условие*, като в зависимост от неговото изпълнение или неизпълнение се извършват различни обработки. Това води до *разклоняване* на съответната програма. За програмиране на разклонения се използват операторите *if* и *switch*
- Многократно изпълнение на последователност от една или повече обработки в *цикъл*, като броят на изпълненията зависи от дадено условие. За програмиране на цикли се използват операторите *while*, *do...while* и *for*

Условията се представят чрез *логически изрази*, като резултатът от пресмятането на такъв израз е стойност на типа **boolean**.

Тип **boolean**. Операции за сравнение. Логически операции. Логически изрази

1. Тип **boolean**

Стойностите на типа **boolean** са **true** и **false**, а *операциите*, които могат да се прилагат над тези стойности, са:

- Операции за сравнение == и !=, чийто резултат е от тип **boolean**
- Операция отрицание !
- Операции конюнкция &, сума по модул две ^ и дизюнкция |
- Операции условна конюнкция && и условна дизюнкция ||
- Условна операция ?:
- Операция присвояване =
- Операция за конкатениране на низове +

Операцията за конкатениране на низове се изпълнява в случай, че единият от операндите е от тип **String**, а другият е от тип **boolean**. Стойността на операнда от тип **boolean** се превръща в низ “true” или “false”, съгласно т. 6.2.5 от Тема 1 и резултатът е конкатенацията на двата низа.

Константите от тип **boolean** са **true** и **false**, а *променлива* е от тип **boolean**, ако в декларацията ѝ е зададен този тип. Инициализаторът на такава променлива е *логически израз*.

Класът **Boolean** от пакета **java.lang** предлага още възможности за обработка на булеви стойности чрез декларираните в него конструктори, методи и константи - виж <http://java.sun.com/javase/6/docs/api/>.

За *изход* на булеви стойности може да се използва методът **System.out.print(Параметър)** от класа **System** на пакета **java.lang**, който преобразува стойността на параметъра в стойност на типа **String** и я извежда в прозорец на экрана.

2. Операции за сравнение на числови стойности

Всяка от операциите за числово сравнение <, <=, >, >=, == и != има вида *ОперацияB*. По време на транслация се получава грешка, ако двата операнда A и B не са от числов тип. По

време на изпълнение двета операнда се превръщат в един и същ тип, съгласно т. 6.2.5 от Тема 1. Ако двета операнда са от целочислен тип, се изпълнява съответната целочислена операция. В противен случай се изпълнява съответната операция за типове с плаваща точка. Резултатът е от тип **boolean**. Таблица 1 съдържа резултатите от изпълнението на операциите за сравнение за различни стойности на A и B.

A	B	A<B	A<=B	A>B	A>=B	A==B	A!=B
5	2	false	false	true	true	false	true
NaN	NaN	false	false	false	false	false	true
-0.0	0.0	false	true	false	true	true	false

Таблица 1. Резултати на операции за числово сравнение

Следващата програма илюстрира особеностите на сравняването на числа с плаваща точка:

```
public class Root {
    public static void main(String[] args) {
        double r=Math.sqrt(2);
        System.out.println("sqrt(2) squared is not 2 but "+(r*r));
    }
}
```

Програмата извежда

sqrt(2) squared is not 2 but 2

Ако се използва `Console.out.printf("% .16f", r*r)`, програмата ще изведе резултата с по-голяма точност, а именно

sqrt(2) squared is not 2 but 2.0000000000000004

При аритметика с плаваща точка резултатът се закръглява до най-близкото число с плаваща точка. Методът `compareDoubles` на класа **Numeric** от пакета `ccj` сравнява две стойности x и y на типа **double** и връща: 0, ако $x=y$, 1, ако $x>y$ и -1, ако $x<y$.

```
public static int compareDoubles(double x, double y) {
    double comp = (x - y) / (1 + Math.max(Math.abs(x), Math.abs(y)));
    if (comp < -EPSILON) return -1;
    if (comp > EPSILON) return 1;
    return 0;
}

private static final double EPSILON = 1E-12;
```

3. Логически операции. Логически изрази

Логическите операции `^`, `||` и `&&` имат вида *A* **Операция** *B*. Операндите A и B трябва да са от тип **boolean**, в противен случай се получава грешка по време на транслация. По време на изпълнение се изчисляват съответно сумата по модул две, дизюнкцията и конюнкцията на двета операнда, като десният операнд на операцията `&&` се изчислява само, ако стойността на левия операнд е `true`, а на операцията `||` - само, ако стойността на левия операнд е `false`.

Операциите `|` и `&` имат същия вид. Операндите трябва да са от тип **boolean**, в противен случай се получава грешка по време на транслация. По време на изпълнение се изчисляват съответно дизюнкцията и конюнкцията на двета операнда.

Логическата операция `!` има вада *A*. Операндът A трябва да е от тип **boolean**, в противен случай се получава грешка по време на транслация. По време на изпълнение се изчислява отрицанието на операнда.

Логическите изрази се съставят и пресмятат съгласно спецификацията на езика Java. *Редът на изпълнение* на операциите може да се укаже явно чрез използване на скоби. Ако няма такива, операциите се изпълняват съгласно техния приоритет от Тема 1. Всички бинарни операции с един и същ приоритет, се изпълняват отляво надясно. Операндите на бинарна операция се изчисляват отляво надясно и след това се изпълнява самата операция.

Задача: Да се състави програма, която извежда true, ако дадена година е високосна и false в противен случай.

Решение: Високосна се нарича година с 366 дни (вместо 365), тоест година, съдържаща 29 февруари. От въвеждането на [григорианския календар](#):

- Годините, кратни на 4 са високосни, останалите не са
- Изключение 1: годините, кратни на 100 не са високосни
- Изключение 2: годините, кратни на 400 са високосни

[Юлианският календар](#), използван преди григорианския, спазва само първото правило.

Реализация: Програма Example2_1:

```
package ContrStmts;

import java.util.Scanner;

public class Example2_1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Year = ");
        int year = sc.nextInt();
        boolean result = ((year % 4 == 0) && (year % 100 != 0))
            || (year % 400 == 0);
        System.out.println(result);
    }
}
```

Задача: Да се състави програма, която проверява верността на законите на Де Морган:

- $!(x \& y) = (!x) | (!y)$
- $!(x | y) = (!x) \& (!y)$

Решение: Нека $f(x,y) = !(x \& y)$ и $g(x,y) = (!x) | (!y)$. Двата израза са *тъждествени*, което означаваме с $f(x,y)=g(x,y)$, ако за всяка двойка $a,b \in \{\text{true},\text{false}\}$ от стойности на променливите x и y е изпълнено $f(a,b)=g(a,b)$. В противен случай $f(x,y) \neq g(x,y)$.

Реализация: Програма Example2_2:

```
package ContrStmts;

public class Example2_2 {
    public static void main(String[] args) {
        boolean a, b, z = true;
        // a=false, b=false
        a = b = false;
        z = z && !(a && b) == !a || !b;
        // a=false, b=true
        b = true;
        z = z && !(a && b) == !a || !b;
        // a=true, b=true
        a = true;
        z = z && !(a && b) == !a || !b;
        // a=true, b=false
    }
}
```

```

        b = false;
        z = z && !(a && b) == !a || !b;
        System.out.println(z);
    }
}

```

Изпълнение на програмата:

true

Условни оператори

1. Оператор if

Пример: Използване на оператор if за програмиране на съставни условия. Нека number е целочислена променлива и $number > 0$.

1. След изпълнението на следващия програмен фрагмент стойността на булевата променлива result е true, ако стойността на number е четирицифрене число и false - в противен случай.

Вариант 1: Използване на операции за сравнение и вложени оператори if

```

if (number>=1000) {
    if(number<=9999) {
        result=true;
    }
    else {
        result=false;
    }
}
else {
    result=false;
}

```

Вариант 2: Използване на логически израз и оператор if

```

if(number>=1000&&number<=9999) {
    result=true;
}
else {
    result=false;
}

```

Вариант 3: Използване на логически израз и оператор за присвояване

```
result=number>=1000&&number<=9999;
```

2. След изпълнението на следващия програмен фрагмент стойността на булевата променлива result е true, ако стойността на number не е четирицифрене число и false - в противен случай.

Вариант 1: Използване на операции за сравнение и вложени оператори if

```

if (number<1000) {
    result=true;
}
else if(number>9999) {
    result=true;
}
else {
    result=false;
}

```

Вариант 2: Използване на логически израз и оператор if

```

if(number<1000 || number>9999) {
    result=true;
}
else {
    result=false;
}

```

Вариант 3: Използване на логически израз и оператор за присвояване

```
result=number<1000 || number>9999;
```

Задача: Да се състави програма, която определя реалните корени на дадено квадратно уравнение $ax^2 + bx + c = 0$, $a \neq 0$.

Реализация: Програма Example2_3:

```
package ContrStmts;
```

```

import java.util.Scanner;

public class Example2_3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a != 0: ");
        double a = sc.nextDouble();
        if (a == 0)
            System.out.println("Incorrect a=0. Try again!");
        else {
            System.out.print("Enter b: ");
            double b = sc.nextDouble();
            System.out.print("Enter c: ");
            double c = sc.nextDouble();
            double D = b * b - 4 * a * c;
            double x1, x2;
            if (D < 0)
                x1 = x2 = Double.NaN;
            else {
                double w;
                x1 = x2 = -b;
                w = 2 * a;
                if (D == 0)
                    x1 = x2 /= w;
                else {
                    D = Math.sqrt(D);
                    x1 = (x1 - D) / w;
                    x2 = (x2 + D) / w;
                }
            }
            System.out.println("x1 = " + x1);
            System.out.println("x2 = " + x2);
        }
    }
}

```

Изпълнение на програмата:

Първо изпълнение:

```
Enter a != 0: 0
Incorrect a = 0. Try again!
```

Трето изпълнение:

```
Enter a != 0: 1
Enter b: -2
Enter c: 1
x1 = 1.0
x2 = 1.0
```

Второ изпълнение:

```
Enter a != 0: 1
Enter b: 1
Enter c: 1
NaN
```

Четвърто изпълнение:

```
Enter a != 0: 1
Enter b: -4
Enter c: -5
x1 = -1.0
x2 = 5.0
```

Задача: Да се състави програма, която чете координатите на три точки $P_1(x_1, y_1)$, $P_2(x_2, y_2)$ и $P_3(x_3, y_3)$ и проверява дали те са върхове на триъгълник. Ако не са, извежда подходящо съобщение. В противен случай определя неговия вид - равностранен, равнобедрен или друг.

Упътване: Трите точки са върхове на триъгълник, ако не лежат на една права. Уравнението $(x-x_1)(y_2-y_1)-(y-y_1)(x_2-x_1)=0$ се нарича *уравнение на правата*, определена от точките $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$. Точката $P_3(x_3, y_3)$ лежи на тази права, ако $(x_3-x_1)(y_2-y_1)-(y_3-y_1)(x_2-x_1)=0$.

$y_1)(x_2 - x_1) = 0$. В противен случай тя не лежи на правата и трите точки са върхове на триъгълник.

Реализация: Програма Example2_4:

```
package ContrStmts;

import java.util.Scanner;

public class Example2_4 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter x1: ");
        double x1 = sc.nextDouble();
        System.out.print("Enter y1: ");
        double y1 = sc.nextDouble();
        System.out.print("Enter x2: ");
        double x2 = sc.nextDouble();
        System.out.print("Enter y2: ");
        double y2 = sc.nextDouble();
        System.out.print("Enter x3: ");
        double x3 = sc.nextDouble();
        System.out.print("Enter y3: ");
        double y3 = sc.nextDouble();
        if ((x3 - x1) * (y2 - y1) - (y3 - y1) * (x2 - x1) == 0)
            System.out.println("Incorrect data. Try again!");
        else {
            double a = (x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1);
            double b = (x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3);
            double c = (x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3);
            if (a == b && b == c)
                System.out.println("Ravnostranen.");
            else if (a == b || a == c || b == c)
                System.out.println("Ravnobedren.");
            else
                System.out.println("Proizvolen.");
        }
    }
}
```

Изпълнение на програмата:

Първо изпълнение:

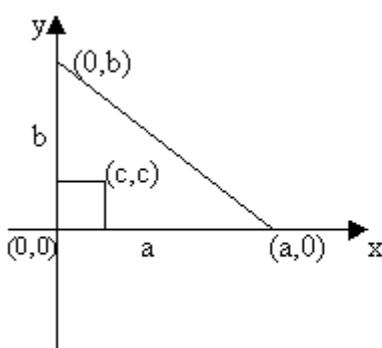
```
Enter x1: 0
Enter y1: 0
Enter x2: 3
Enter y2: 0
Enter x3: 0
Enter y3: 3
Ravnobedren.
```

Второ изпълнение:

```
Enter x1: 0
Enter y1: 0
Enter x2: 5
Enter y2: 0
Enter x3: 0
Enter y3: 1
Proizvolen.
```

Задача: Да се състави програма, която по зададени катети а и б на правоъгълен триъгълник и точка с координати X и Y, съгласно избраната координатна система на рисунката по-долу, проверява дали точката лежи вътре в триъгълника и всяко от разстоянията ѝ до катетите е най-малко с.

Решение:



Стойностите на a и b трябва да са положителни числа, а на c - да е такова, че точката $P(c,c)$ да лежи в триъгълника.

Уравнението на правата $L(x,y)$, минаваща през точките $P_1(0,b)$ и $P_2(a,0)$ е $L(x,y)=bx+ay-ab=0$.

За всяка точка $P_3(M,N)$, лежаща в една и съща полуправнина с точката $P_0(0,0)$, $L(M,N)$ и $L(0,0)$ имат един и същ знак. От $L(0,0)=-ab<0$ следва, че $L(c,c)=bc+ac-ab<0$. Следователно, c трябва да е такова, че $0 < c < (ab)/(a+b)$.

Следователно, точка с координати $P(X,Y)$ отговаря на условието, ако $X>=c$, $Y>=c$ и $aY+bX-ab<0$.

Реализация: Програма Example2_5:

```
package ContrStmts;

import java.util.Scanner;

public class Example2_5 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a>0: ");
        double a = sc.nextDouble();
        System.out.print("Enter b>0: ");
        double b = sc.nextDouble();
        if (a <= 0 || b <= 0)
            System.out.println("Invalid data a<=0 or b<=0. Try again!");
        else {
            double r = (a * b) / (a + b);
            System.out.print("Enter 0<c<" + r + ": ");
            double c = sc.nextDouble();
            if (c <= 0 || c >= r)
                System.out.println("Incorrect c = " + c + ". Try again!");
            else {
                System.out.print("Enter x: ");
                double x = sc.nextDouble();
                System.out.print("Enter y: ");
                double y = sc.nextDouble();
                if (x >= c && y >= c && (a * y + b * x - a * b) < 0)
                    System.out.println("Yes");
                else
                    System.out.println("No");
            }
        }
    }
}
```

Изпълнение на програмата:

Първо изпълнение:

```
Enter a>0: 3.0
Enter b>0: 5.0
Enter 0<c<1.875: 1.0
Enter x: 2.0
Enter y: 2.0
No
```

Второ изпълнение:

```
Enter a>0: 10.0
Enter b>0: 25.0
Enter 0<c<7.142857142857143: 2.0
Enter x: 3.0
Enter y: 4.0
Yes
```

2. Оператор switch

Пример: Нека day е целочислена променлива.

1. Изпълнението на следващия програмен фрагмент се състои в извеждане на името на деня, ако $day \in \{1,2,3,4,5,6,7\}$. В противен случай се извежда Incorrect day!.

Вариант 1: Използване на оператор if

```
if(day==1)
    System.out.println("Monday");
else if(day==2)
    System.out.println("Tuesday");
else if(day==3)
    System.out.println("Wednesday");
else if(day==4)
    System.out.println("Thursday");
else if(day==5)
    System.out.println("Friday");
else if(day==6)
    System.out.println("Saturday");
else if(day==7)
    System.out.println("Sunday");
else
    System.out.println("Incorrect day");
```

Вариант 2: Използване на оператор switch

```
switch (day) {
    case 1: System.out.println("Monday"); break;
    case 2: System.out.println("Tuesday"); break;
    case 3: System.out.println("Wednesday"); break;
    case 4: System.out.println("Thursday"); break;
    case 5: System.out.println("Friday"); break;
    case 6: System.out.println("Saturday"); break;
    case 7: System.out.println("Sunday"); break;
    default: System.out.println("Incorrect day!"); break;
}
```

2. Изпълнението на следващия програмен фрагмент се състои в извеждане на Workday, ако $day \in \{1,2,3,4,5\}$. В противен случай се извежда Weekend, ако $day \in \{6,7\}$. В противен случай се извежда Incorrect day!.

Вариант 1: Използване на оператор if

```
if(day==1 || day==2 || day==3 || day==4 || day==5)
    System.out.println("Workday");
else if(day==6 || day==7)
    System.out.println("Weekend");
else
    System.out.println("Incorrect day");
```

Вариант 2: Използване на оператор switch

```
switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5: System.out.println("Workday"); break;
    case 6:
    case 7: System.out.println("Weekend"); break;
    default: System.out.println("Incorrect day!"); break;
}
```

Задача: Да се състави програма, която чете цяло число $n > 0$ и число $value > 0$, представляващо количество в съответната на п лява мерна единица на дадената таблица. Ако входните данни са коректни, определя и извежда количеството в съответната дясна мерна единица. В противен случай извежда подходящо съобщение.

n	
1	1 унция (за течности) = 29.586 милилитра
2	1 галон = 3.785 литра
3	1 унция = 28.3495 грама
4	1 паунд = 453.6 грама
5	1 инч = 2.54 сантиметра
6	1 фут = 30.5 сантиметра
7	1 миля = 1.609 километра

Реализация: Програма Example2_6:

```
package ContrStmts;

import java.util.Scanner;

public class Example2_6 {
    public static void main(String[] args) {
```

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter 1<=n<=7: ");
int n = sc.nextInt();
if (n < 1 || n > 7)
    System.out.println("Incorrect n = " + n + ". Try again!");
else {
    System.out.print("Enter value>0: ");
    double value = sc.nextDouble();
    if (value <= 0)
        System.out.println("Incorrect value = " + value
                           + ". Try again!");
    else {
        switch (n) {
            case 1:
                value *= 29.586;
                break;
            case 2:
                value *= 3.785;
                break;
            case 3:
                value *= 28.3495;
                break;
            case 4:
                value *= 453.6;
                break;
            case 5:
                value *= 2.54;
                break;
            case 6:
                value *= 30.5;
                break;
            case 7:
                value *= 1.609;
                break;
        }
        System.out.println("value = " + value);
    }
}
```

3. Условна операция ?:

Пример: Нека x и y са числови променливи. След изпълнението на следващия програмен фрагмент стойността на числовата променлива result е равна на по-малката от стойностите на x и y.

Вариант 1: Използване на оператор if

```
if(x<y) {  
    result=x;  
}  
else {  
    result=y;  
}
```

Вариант 2: Използване на условна операция ?:

```
result=x<y ? x : y;
```

Условната операция ?: има вида $A ? B : C$. Първият операнд А трябва да е логически израз, а вторият В и третият С - трябва да са едновременно логически изрази, аритметични изрази или от съставен тип. В противен случай се получава грешка по време на транслация. Типът на резултата се определя от типа на вторите два операнда по следния начин: ако те са от един и същ тип, то това е типът на резултата. В противен случай, ако те са от числов тип, то:

1. Ако единият операнд е от тип **byte**, а другият – от тип **short**, то типът на резултата е **short**
2. В противен случай, ако единият от operandите е от тип T, където T е **byte**, **short** или **char**, а другият operand е константен израз от тип **int**, чията стойност може да се представи като стойност на типа T, то типът на резултата е типът T
3. В противен случай двата operandи се превръщат в един и същ тип, съгласно т.6.2.5 от Тема 1 и това е типът на резултата

По време на изпълнение първо се изчислява стойността на първия operand и ако тя е true, се изчислява вторият operand. В противен случай се изчислява третият operand. Получената стойност се превръща в стойност на типа на резултата и преобразуваната стойност е резултатът на операцията.

Оператори за цикъл while, do...while и for

Задача: Да се състави програма, която чете редица от n реални числа и определя средното аритметично на първите $m \leq n$ положителни числа в редицата.

Реализация: Програма Example2_7:

```
package ContrStmts;

import java.util.Scanner;

public class Example2_7 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter n>0: ");
        int n = sc.nextInt();
        if (n <= 0)
            System.out.println("Incorrect n = " + n + ". Try again!");
        else {
            System.out.print("Enter 0<m<=n: ");
            int m = sc.nextInt();
            if (m <= 0 || m > n)
                System.out.println("Incorrect m = " + m + ". Try again!");
            else {
                double sum = 0;
                int count = 0;
                for (int i = 0; i < n; i++) {
                    System.out.print("Enter a number: ");
                    double a = sc.nextDouble();
                    if (a <= 0)
                        continue;
                    count++;
                    if (count <= m)
                        sum += a;
                    if (count == m)
                        break;
                }
                System.out.println(count + " positive numbers");
                if (count != 0) {
                    double average = sum / count;
                    System.out.println("The average is " + average);
                }
            }
        }
    }
}
```

Изпълнение на програмата:

Първо изпълнение:

```
Enter n>0: 3
Enter 0<m<=n: 3
Enter a number: -1
Enter a number: 0
Enter a number: -2
0 positive numbers
```

Второ изпълнение:

```
Enter n>0: 7
Enter 0<m<=n: 3
Enter a number: 1
Enter a number: -1
Enter a number: 2
Enter a number: 3
3 positive numbers
The average is 2.0
```

Трето изпълнение:

```
Enter n>0: 7
Enter 0<m<=n: 3
Enter a number: -1
Enter a number: 1
Enter a number: -2
Enter a number: -3
Enter a number: 2
Enter a number: -4
Enter a number: -5
2 positive numbers
The average is 1.5
```

Задача: Да се състави програма, която определя най-големия общ делител на две цели положителни числа.

Алгоритъм на Евклид: Дадени са целите числа A>0 и B>0. Като резултат се получава техният най-голям общ делител НОД(A,B).

1. $P = A \% B$
2. Ако $P = 0$, край на алгоритъм и $\text{НОД}(A,B)=B$. Иначе – переход на т. 3
3. $A = B$
4. $B = P$
5. $P = A \% B$
6. Преход на т. 2

Реализация: Програма Example2_8:

```
package ContrStmts;

import java.util.Scanner;

public class Example2_8 {
    public static void main(String[] args) {
        int A, B, C, D;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter an integer A>0: ");
        A = C = sc.nextInt();
        System.out.print("Enter an integer B>0: ");
        B = D = sc.nextInt();
        if (A <= 0 || B <= 0)
            System.out.println("Invalid data: A<=0 or B<=0. Try again!");
        else {
            int P = A % B;
            while (P != 0) {
                A = B;
                B = P;
                P = A % B;
            }
            System.out.println("GCD(" + C + "," + D + ") = " + B);
        }
    }
}
```

Изпълнение на програмата:

Първо изпълнение:

```
Enter an integer A>0: 12
Enter an integer B>0: 8
GCD(12,8) = 4
```

Второ изпълнение:

```
Enter an integer A>0: 150
Enter an integer B>0: 111
GCD(150,111) = 3
```

Трето изпълнение:

```
Enter an integer A>0: 139
Enter an integer B>0: 37
GCD(139,37) = 1
```

Задача: Да се състави програма, която проверява дали естествено число съвпада с неговото обратно число.

Решение: Нека $N=d_n\dots d_0$ е естествено число. Числото $M=d_0\dots d_n$ се нарича *обратно число* на N . Числото $M=d_010^n+\dots+d_n=(\dots((d_010+d_1)10+d_2)10+\dots+d_{n-1})10+d_n$, съгласно правилото на Хорнер за пресмятане на стойност на полином. Цифрите на числото N се отделят, прилагайки алгоритъма чрез деление за преобразуване на десетично число в р-ична позиционна бройна система (в случая $p=10$).

Реализация: Програма Example2_9:

```
package ContrStmts;

import java.util.Scanner;

public class Example2_9 {
    public static void main(String[] args) {
        int N, N1, p = 10;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a positive integer number: ");
        N = N1 = sc.nextInt();
        if (N <= 0)
            System.out.println("Incorrect N = " + N + ". Try again!");
        else {
            int M = 0, d;
            do {
                d = N % p;
                M = M * p + d;
                N /= p;
            } while (N != 0);
            System.out.println("The reverse number of " + N1 + " is " + M);
            System.out.println(N1 == M ? "Yes" : "No");
        }
    }
}
```

Изпълнение на програмата:

```
Enter a positive integer number: 123
The reverse number of 123 is 321
No
```

Задача: Нека $a_0>0$ е цяло число. Редицата от цели числа $a_0, a_1, \dots, a_i, a_{i+1}, \dots$ се получава по следния начин: ако a_i е нечетно число, то $a_{i+1} = 1+3a_i$. В противен случай a_i се дели на 2 до получаване на нечетно число b и $a_{i+1} = b$. Да се състави програма, която за $a_0>1$ и $n\geq 2$ проверява дали сред членовете на редицата $a_0, a_1, a_2, \dots, a_n$ има двойка съседни членове (1,4).

Реализация: Програма Example2_10:

```
package ContrStmts;

import java.util.Scanner;

public class Example2_10 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter an integer a0>1: ");
        int a = sc.nextInt();
        if (a <= 1)
            System.out.println("Incorrect a0 = " + a + ". Try again!");
        else {
```

```
System.out.print("Enter an integer n>=2: ");
int n = sc.nextInt();
if (n < 2)
    System.out.println("Incorrect n = " + n + ". Try again!");
else {
    for (int i = 1; i < n; i++) {
        if (a % 2 != 0)
            a = 1 + 3 * a;
        else
            do
                a /= 2;
            while (a % 2 == 0);
        if (a == 1)
            break;
    }
    System.out.println(a == 1 ? "Yes" : "No");
}
}
```



Задачи за упражнение

Задача: Да се състави програма, която за всяка двойка стойности на булевите променливи А и В извежда техните отрицания, конюнкция, дизюнкция и сума по модул две. Изпълнете програмата и попълнете получените резултати в Таблица 1.

A	B	!A	!B	A & B	A B	A ^ B
false	false					
false	true					
true	false					
true	true					

Таблица 1. Резултати на операциите отрицание, конюнкция, дизюнкция и сума по модул две

Задача: Да се промени програмата, която по три страни на триъгълник определя неговите периметър, лице и единия от ъглите, като това се прави само, ако числата са страни на триъгълник и освен това се определя и неговият вид.

Задача: Да се състави програма, която за дадено четиризначно число проверява дали

- Числото е четно
 - Квадратът на числото е равен на сумата от кубовете на неговите цифри
 - Сумата от неговите цифри е двузначно число

Задача: Да се състави програма, която за дадено четиризначно число проверява дали

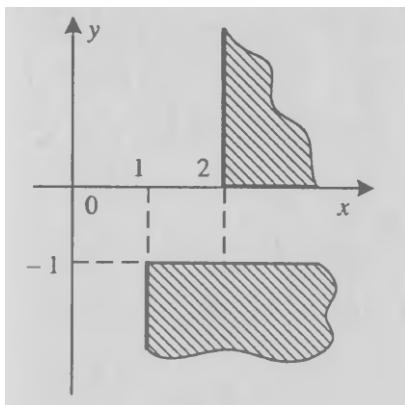
- Числото съдържа цифрата 3 или 7
 - Числото съдържа цифрата 6 поне веднъж
 - Числото съдържа цифрата 5 и то само веднъж

Задача: Да се състави програма, която за дадено четиризначно число проверява дали

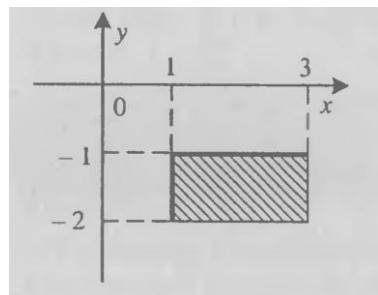
- Числото съдържа поне две еднакви цифри
 - В записа на числото цифрите са наредени в нарастващ ред (отляво надясно)

Задача: Да се състави програма, която проверява дали от три топа на шахматна дъска 8 x 8 има поне два, които се бият.

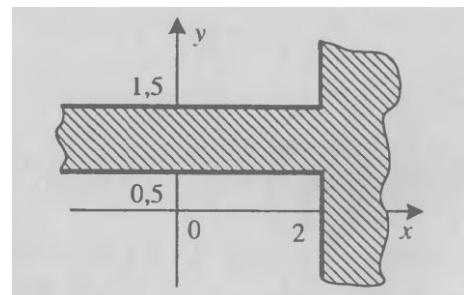
Задача: Да се състави програма, която проверява дали дадена точка $P(x,y)$ принадлежи на заштрихованата област:



a)



б)



в)

Задача: Дата дд.мм.гггг може да се представи като осемцифрен число ддммгггг. Да се състави програма, която за дадена дата:

- Проверява дали годината е високосна
- Проверява на кой сезон принадлежи денят, използвайки следващата таблица

SEASON	FROM	TO
Spring	March 21	June 20
Summer	June 21	September 22
Fall	September 23	December 21
Winter	December 21	March 20

- Проверява дали денят е работен или почивен

Задача: Да се състави програма, която по дата на раждане (година, номер на месец и ден) на човек и по година, номер на месец и номер на ден на текущата дата:

- Определя възрастта на човека
- Определя зодията, на която принадлежи денят, използвайки следващата таблица

SIGN	FROM	TO
Capricorn	December 22	January 19
Aquarius	January 20	February 17
Pisces	February 18	March 19
Aries	March 20	April 19
Taurus	April 20	May 20
Gemini	May 21	June 20
Cancer	June 21	July 22
Leo	July 23	August 22
Virgo	August 23	September 22
Libra	September 23	October 22
Scorpio	October 23	November 21
Sagittarius	November 22	December 21

Задача: Да се състави програма, която проверява дали две дадени прости прави в равнината се пресичат и извежда подходящо съобщение за резултата на проверката. Освен това, ако правите се пресичат, определя и извежда координатите на пресечната им точка.

Задача: Да се състави програма, която проверява дали две естествени числа са взаимно прости.

Задача: Да се състави програма, която проверява дали естествено число е просто.

Задача: Да се състави програма, която определя първите n съвършени числа. Естествено число е „съвършено”, ако е равно на сумата от делителите си (включително 1), например 28: $28=1+2+4+7+14$.

Задача: Естествени числа A и B са „приятелски”, ако сумата от делителите на A е равна на B и обратно. Да се състави програма, която определя всички двойки приятелски числа, принадлежащи на даден интервал.

Задача: Да се състави програма, която определя първите n автоморфни числа. Естествено число е „автоморфно”, ако се съдържа в края на своя квадрат, например 25: $25^2=625$.

Задача: Да се състави програма, която изпитва дете първо за таблициата за събиране и след това – за таблицата за умножение на числата от 1 до 10, като резултатът е най-много 10. За всяка от таблиците определя и извежда оценка по следния начин: за всеки грешен отговор от 6.00 вади 0.10, като при 40 и повече грешни отговора оценката е 2.00.

Задача: Да се състави програма, която за n дадени точки $P(x_1,y_1), \dots, P(x_n,y_n)$ определя броя на точките, които принадлежат на пръстена, образуван от две дадени концентрични окръжности.

Задача: Да се състави програма, която извежда таблица, съдържаща температурата по Фаренхайт $F \in \{0, 20, 40, \dots, 300\}$ и нейния еквивалент по Целзий C, който се определя по формулата $C=5/9(F-32)$.

Задача: Методът на трапеците за приближено пресмятане на лицето на фигура, получена чрез функцията $f(x)$ и прави през точките $A=(a, f(a))$ и $B=(b, f(b))$, успоредни на оста y, се състои в разделянето на интервала $[a;b]$ на m подинтервала с една и съща дължина. Приближената стойност на лицето се определя по формулата

$$S = [f(a) + \sum_{k=1}^{m-1} (f(a+kh) + f(b)], h = (b-a)/m$$

Да се състави програма, която за дадени интервал $[a;b]$ и цяло число $m > 0$, пресмята лицето на фигурата, определена от функцията $f(x) = (1-x^2)^{-1/2}$.

Задача: Да се състави програма, която определя всички прости делители на естествено число и техните кратности.

Упътване: Кратност на делител k на естествено число A наричаме най-голямото цяло положително число p, такова че k^p дели A.

Задача: Да се състави програма, която чете за всеки от N на брой студенти факултетен номер и 10 оценки от изпити. Пресмята и извежда:

- За всеки студент – факултетен номер и среден успех
- Брой студенти със среден успех в интервала [3,00; 3,50)
- Брой студенти със среден успех в интервала [3,50; 4,50)
- Брой студенти със среден успех в интервала [4,50; 5,50)
- Брой студенти със среден успех в интервала [5,50; 6,00]

Задача: Да се състави програма, която чете естествено число N.

- Извежда имената на цифрите в осмичния запис на числото от младшата към старшата. Например, за $12 = 14_8$ програмата извежда: четири едно
- Проверява дали всяка от цифрите 1 и 7 се срещат в осмичния запис на числото и то само веднъж



Продедурно програмиране

Прилагането на метода за решаване на приста задача за задачи с повече обработки, води до получаване на програма от един метод, което затруднява нейното тестване и модификация. Това неудобство може да бъде отстранено, като на етапа на проектиране задачата се декомпозира на подзадачи, изяснява се взаимодействието между тях и за всяка една се определя алгоритъм за нейното решаване. Реализацията се състои в определяне на подходящи структури от данни, реализация на всяка подзадача във вид на метод и програмиране на установеното взаимодействие.

Представянето на програмата като колекция от методи, от една страна дава възможност тя да се изгражда стъпка по стъпка, като всеки метод се реализира и тества самостоятелно, след което се интегрира с вече готовите. Това улеснява нейното тестване, защото може да се провери дали полученият вариант на програмата работи добре. От друга страна се улеснява модификацията на програмата чрез промяна на метод, отстраняване на метод, добавяне на нов метод.

Недостатъците произтичат от това, че методите обработват общи структури от данни, което може да доведе до необходимост от промяна на методи при промяна в данните и обратно.

Използването на класове предоставя възможност за разширяване на наличните библиотеки с нови средства.

Декларацията на клас включва:

- Спецификатор за достъп
- Ключовата дума class
- Тяло на клас

Тялото на класа съдържа:

- Декларации на променливи на клас и на екземпляри (за данните на обектите на класа)
- Конструктори за инициализиране на обектите на класа
- Методи на клас и методи на екземпляри (за използване на обектите на класа)
- Метод finalize, който се изпълнява при унищожаването на обектите на класа

Променливите и методите се наричат **членове** на класа.

Декларацията на променлива-член на клас включва:

- Спецификатор за достъп
- Ключовата дума static (променлива на клас)
- Има на тип
- Име на променлива

*Достъп до променлива на клас: **Име_на_клас.Име_на_променлива**.*

Декларацията на метод-член на клас включва:

- Спецификатор за достъп
- Ключовата дума static (метод на клас)
- Има на тип на връщания резултат
- Име на метод
- Списък от параметри

*Извикване на метод на клас: **Име_на_клас.Извикване_на_метод**.*

Следващата таблица показва правото на достъп, разрешено от съответния спецификатор за достъп.

Specifier	class	subclass	package	World
Private	X			
Protected	X	X*	X	
Public	X	X	X	X
Package	X		X	

Името на клас се явва име на съставен тип и може да се използва на всяко място, на което може да се използва име на тип.

Обектите се явяват *екземпляри на класове*. Жизненият цикъл на обект на даден клас включва:

- Създаване на обекта:
Име_на_клас Име_на_обектова_променлива = new Конструктор_на_класа
- Използване на обекта чрез извикване на методите му:
Име_на_обектова_променлива.Извикване_на_метод
- Унищожаване на обекта

Програма Test: Програмата **Test** илюстрира преобразуването при предаването на параметри от примитивен тип.

```
package ClassMethods;

public class Test {
    static int m(byte a, int b) {
        return a + b;
    }

    static int m(short a, short b) {
        return a - b;
    }

    public static void main(String[] args) {
        System.out.println(m(12, 2));
    }
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  The method m(byte, int) in the type Test is not applicable for the arguments
  (int, int)
  at ClassMethods.Test.main(Test.java:13)
```

Стойността на фактическия параметър се присвоява на формалния параметър, при което е допустимо само неявно разширяващо преобразуване. Ако е необходимо стесняващо преобразуване на типа, то се указва явно. По време на транслация на програмата се получава грешка, защото константите 12 и 2 са от тип **int**, който не съответства на типа на формалните параметри. Тя се отстранява, като 12 и 2 се преобразуват явно в типа на съответния формален параметър.

```
package ClassMethods;

public class Test {
    static int m(byte a, int b) {
        return a + b;
    }
```

```

static int m(short a, short b) {
    return a - b;
}

public static void main(String[] args) {
    System.out.println(m((byte) 12, 2));
    System.out.println(m((short) 12, (short) 2));
}
}

```

Изпълнение на програмата:

14
10

Програма Example3_1: Програмата Example3_1 илюстрира предаването на параметри от примитивен тип.

```

package ClassMethods;

public class Example3_1 {

    public static void swap(int arg1, int arg2) {
        int work = arg1;
        arg1 = arg2;
        arg2 = work;
    }

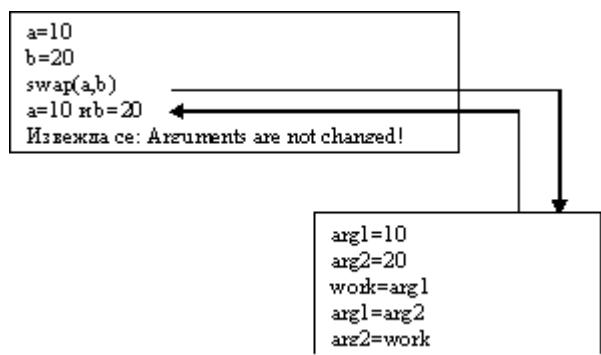
    public static void main(String[] arg) {
        int a = 10, b = 20;
        swap(a, b);
        if (a == 20 && b == 10)
            System.out.println("Arguments are swaped!");
        else if (a == 10 && b == 20)
            System.out.println("Arguments are not changed!");
        else
            System.out.println("Too strange!");
    }
}

```

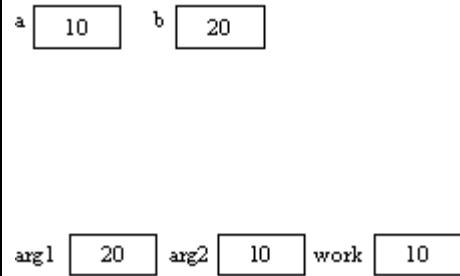
Изпълнение на програмата:

Arguments are not changed!

Проследяване на изпълнението на програмата:



Локални променливи и параметри в ОП:



Задача: Дадена е редицата a_1, a_2, \dots, a_n , $n \geq 2$, от естествени числа. Да се състави програма, която определя броя на съвършените числа в редицата и проверява дали в нея има две различни съседни приятелски числа.

Алгоритъм Solution за решаване на задачата: Дадени са естествените числа $n \geq 2$, a_1, a_2, \dots, a_n . Като резултат се получава броят на съвършените числа count и стойността на променливата friendly: true, ако в редицата има две различни съседни приятелски числа и false – в противен случай.

1. $i = 1$
2. $a = a_1$
3. $count = isPerfect(a) ? 1 : 0$
4. $friendly = false$
5. $i = 2$
6. Ако $i \leq n$, переход на т. 7. Иначе – край на алгоритъма
7. $b = a_i$
8. $count = count + isPerfect(b) ? 1 : 0$
9. Ако $a \neq b$, переход на т. 10. Иначе – переход на т. 12.
10. $friendly = friendly || areFriendly(a,b)$
11. $a = b$
12. $i = i + 1$
13. Преход на т. 6

Използвани са функциите `isPerfect(n)`, която приема стойност `true`, ако n е съвършено число и `false` – в противен случай и `areFriendly(a,b)`, която приема стойност `true`, ако a и b са приятелски числа и `false` – в противен случай.

Алгоритъм isPerfect: Дадено е естественото число n . Като резултат се получава стойността на променливата `result`: `true`, ако n е съвършено число и `false` – в противен случай.

1. `result = (n == sum(n))`

Използвана е функцията `sum(n)` за получаване на сумата от делителите на n .

Алгоритъм areFriendly: Дадени са естествените числа a и b . Като резултат се получава стойността на променливата `result`: `true`, ако a и b са приятелски числа и `false` – в противен случай.

1. `result = (a == sum(b)) && (b == sum(a))`

Използвана е функцията `sum(n)` за получаване на сумата от делителите на n .

Алгоритъм sum: Дадено е естественото число n . Като резултат се получава `result` -сумата от делителите (включително и числото 1) на n .

1. `result = 1`
2. $i = 2$
3. Ако $i < n/2$, переход на т. 4. Иначе – край на алгоритъма
4. Ако $n \% i == 0$, переход на т. 5. Иначе – переход на т. 6
5. `result = result + i`
6. $i = i + 1$
7. Преход на т. 3

Спецификация на програмата:

```
public class Example3_2 {  
    public static int sum(int n) {...}  
    //Определя сумата от делителите (включително и числото 1) на цялото число n  
    public static boolean isPerfect(int n) {...}
```

```
//Проверява дали естественото число n е съвършено
public static boolean areFriendly(int a,int b) {...}
//Проверява дали естествените числа a и b са приятелски
public static void solution() {...}
//Решава задачата.
public static void main(String[] args) {...}
//Извиква метода solution
}
```

Реализация:

```
package ClassMethods;

import java.util.Scanner;

public class Example3_2 {
    public static int sum(int n) {
        int result = 1;
        for (int i = 2; i <= n / 2; i++)
            if (n % i == 0)
                result += i;
        return result;
    }

    public static boolean isPerfect(int n) {
        return n == sum(n);
    }

    public static boolean areFriendly(int a, int b) {
        return a == sum(b) && b == sum(a);
    }

    public static void solution() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter an integer n>=2: ");
        int n = sc.nextInt();
        System.out.print("      Enter a positive integer: ");
        int a = sc.nextInt();
        int count = isPerfect(a) ? 1 : 0;
        boolean friendly = false;
        for (int i = 2; i <= n; i++) {
            System.out.print("      Enter a positive integer: ");
            int b = sc.nextInt();
            count += isPerfect(b) ? 1 : 0;
            if (a != b)
                friendly = friendly || areFriendly(a, b);
            a = b;
        }
        System.out.println("The number of perfect integers is " + count);
        System.out.println(friendly ? "Yes" : "No");
    }

    public static void main(String[] arg) {
        solution();
    }
}
```

Изпълнение:

Първо изпълнение:

```
Enter an integer n>=2: 4
Enter a positive integer: 28
Enter a positive integer: 220
```

Второ изпълнение:

```
Enter an integer n>=2: 3
Enter a positive integer: 220
Enter a positive integer: 28
```

```
Enter a positive integer: 284      Enter a positive integer: 284
Enter a positive integer: 6          The number of perfect integers is 1
The number of perfect integers is 2      No
Yes
```

Задача: Дадена е редицата a_1, a_2, \dots, a_n , $n \geq 1$, от реални неотрицателни числа. Да се състави програма, която определя корен квадратен от всеки член на редицата.

Решение: Нека $\arg \geq 0$ и $E > 0$ са реални числа. Методът на Нютон за пресмятане на *корен квадратен от arg с точност E* се състои в следното: генерира се редицата от реални числа, $y_0, y_1, \dots, y_{k-1}, y_k, \dots$ съгласно формулите:

$$\begin{cases} y_0 = 1 \\ y_k = (y_{k-1} + \arg / y_{k-1})/2, \text{ за } k \geq 1 \end{cases}$$

до получаване на двойка съседни членове y_p и y_{p+1} , такива че $|y_p - y_{p+1}| < E$. Числото y_{p+1} е корен квадратен от \arg с точност E .

Спецификация на програмата:

```
public class Example3_3 {
    public static double E = 1E-12;
    //Декларация и инициализация на променлива на клас E
    public static double sqrt(double arg) {...}
    //Пресмята корен квадратен от arg с точност E по метода на Нютон
    public static void testSqrt() {...}
    //Чете n неотрицателни реални числа и за всяко реално число x пресмята и извежда
    //sqrt(x) и Math.sqrt(x) с цел проверка, че корен квадратен се пресмята правилно от
    //метода sqrt
    public static void main(String[] args) {...}
    //Извиква метода testSqrt
}
```

Реализация:

```
package ClassMethods;

import java.util.Scanner;

public class Example3_3 {
    public static double E = 1E-12;

    public static double sqrt(double x) {
        if (x < 0)
            return Double.NaN;
        double y0, y1 = 1;
        do {
            y0 = y1;
            y1 = (x / y0 + y0) / 2.0;
        } while (Math.abs(y0 - y1) >= E);
        return y1;
    }

    public static void testSqrt() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter n>0: ");
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            System.out.print("Enter x>0: ");
            double x = sc.nextDouble();
            System.out.println("sqrt(x) = " + sqrt(x));
        }
    }
}
```

```

        System.out.println("      Math.sqrt(x) = " + Math.sqrt(x));
    }

    public static void main(String[] args) {
        testSqrt();
    }
}

```

Задача: Да се реализира клас **MyIntegers** на пакета **Library**, който предоставя методи за обработка на цели числа, съгласно спецификация:

```

package Library;

public class MyIntegers {
    public static int gcd(int a,int b) {...}
    //Определя най-големия общ делител на целите числа a и b
    public static boolean isTrue(int a,int b) {...}
    //Проверява дали целите числа a и b са взаимно прости
    public static boolean isEven(int n) {...}
    //Проверява дали цялото число n е четно
    public static int sum(int n) {...}
    //Определя сумата от делителите на цялото число n
    public static boolean isPerfect(int n) {...}
    //Проверява дали естественото число n е съвършено
    public static boolean areFriendly(int a,int b) {...}
    //Проверява дали естествените числа a и b са приятелски
    public static boolean isPrime(int n) {...}
    //Проверява дали цялото число n е просто
    public static int numberOfDigits(int n) {...}
    //Определя броя на цифрите на цялото число n
    public static int sumOfDigits(int n) {...}
    //Определя сумата от цифрите на цялото число n
    public static int reverseNumber(int n) {...}
    //Определя обратното число на цялото число n
    public static boolean isPolindrom(int n) {...}
    //Проверява дали цялото число n е палиндром (съвпада със своето обратно число)
    public static int numberOfDigits(int n,int p) {...}
    //Определя броя на цифрите на р-ичния запис на цялото число n
    public static int sumOfDigits(int n,int p) {...}
    //Определя сумата от цифрите на р-ичния запис на цялото число n
}

```

Реализация:

```

package Library;

public class MyIntegers {
    public static int gcd(int a, int b) {
        System.out.println("Not implemented!");
        return -1;
    }

    public static boolean isTrue(int a, int b) {
        System.out.println("Not implemented!");
        return true;
    }

    public static boolean isEven(int n) {
        System.out.println("Not implemented!");
        return true;
    }

    public static int sum(int n) {

```

```

int result = 1;
for (int i = 2; i <= n / 2; i++)
    if (n % i == 0)
        result += i;
return result;
}

public static boolean isPerfect(int n) {
    return n == sum(n);
}

public static boolean areFriendly(int a, int b) {
    return a == sum(b) && b == sum(a);
}

public static boolean isPrime(int n) {
    System.out.println("Not implemented!");
    return true;
}

public static int numberOfDigits(int n) {
    System.out.println("Not implemented!");
    return -1;
}

public static int sumOfDigits(int n) {
    System.out.println("Not implemented!");
    return -1;
}

public static int reverseNumber(int n) {
    System.out.println("Not implemented!");
    return -1;
}

public static boolean isPolindrom(int n) {
    System.out.println("Not implemented!");
    return true;
}

public static int numberOfDigits(int n, int p) {
    System.out.println("Not implemented!");
    return -1;
}

public static int sumOfDigits(int n, int p) {
    System.out.println("Not implemented!");
    return -1;
}
}

```

Задача: Да се реализират:

1. Клас **MyMath** на пакета **Library**, съгласно спецификация:

```
package Library;
```

```

public class MyMath {
    public static double E=1E-12;
    //Променлива за точността на изчисленията
    public static double E() {...}
    //Пресмята числото e с точност E
    public static double PI() {...}
    //Пресмята числото π с точност E
    public static double sqrt(double arg) {...}
    //Пресмята корен квадратен от arg с точност E
}

```

```

public static double exp(double arg) {...}
//Пресмята earg с точност E
public static double sin(double arg) {...}
//Пресмята sin(arg) с точност E
public static double cos(double arg) {...}
//Пресмята cos(arg) с точност E
public static double arctg(double arg) {...}
//Пресмята arctg(arg) с точност E
}

```

2. Клас **TestMyMath** на пакета **Tests**, съдържащ по един метод за тестване за всеки от методите на класа **MyMath** и main метод, който извиква методите за тестване

Решение: Ще използваме, че:

$1. \ e = \sum_{k=0}^{\infty} \frac{1}{k!}$	$4. \ \sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{(2k+1)}}{(2k+1)!}, \quad x \in (-\infty, +\infty)$
$2. \ \pi = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} \frac{4}{(2k+1)}$	$5. \ \cos(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}, \quad x \in (-\infty, +\infty)$
$3. \ e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}, \quad x \in (-\infty, +\infty)$	$6. \ \arctg(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{(2k+1)}}{2k+1}, \quad -1 \leq x \leq 1$

и това, че във всеки един от случаите от т.1 до т.6, за всяко реално число $E > 0$ може да се определи сумата $\text{sum} = t_1 + t_2 + \dots + t_p$ от реални числа t_1, t_2, \dots, t_p такива, че $|t_i| \geq E$ за всяко $i \in [1; p-1]$ и $|t_p| < E$. Това число е *стойността с точност E* на е от т.1, на π от т.2 и на стойността на функцията за дадена стойност на променливата за всяка от останалите точки.

За определяне на сумата sum с точност $E > 0$ може да се използва следният шаблон:

```

double term; //Променлива за текущото събирамо
double sum; //Променлива за сумата
term = първото събирамо;
sum = term;

while(Math.abs(term) >= E) {
    //На term се присвоява следващото събирамо, като при неговото пресмятане се
    //използва текущото събирамо или негови части, ако е възможно
    sum = sum + term;
}

```

Реализация: Предложената реализация включва само част от методите, като методите sqrt и testSqrt от предишната задача се включват съответно в **MyMath** и **TestMyMath**.

1. Метод main на класа **TestMyMath**
2. Пресмятане на числото π с точност E
 - 2.1.Метод PI на класа **MyMath**
 - 2.2.Метод testPI на класа **TestMyMath**, който чете последователност от стойности за точността на пресмятанията и за всяко число epsilon от нея прави следното. Ако $0 < \text{epsilon} < 1$ определя точността на пресмятането $\text{MyMath.E} = \text{epsilon}$ и извежда $\text{MyMath.PI}()$, Math.PI и ако $|\text{MyMath.PI}() - \text{Math.PI}| < E$, извежда Yes и No в противен случай. Ако $\text{epsilon} \leq 0$ или $\text{epsilon} \geq 1$, изпълнението на метода се прекратява.
 3. Пресмятане на e^{arg} за $x = \text{arg}$ с точност E

3.1. Метод `exp` на класа **MyMath**

3.2. Метод `testExp` на класа **TestMyMath**, който чете $n \geq 2$ реални числа x_1, \dots, x_n и пресмята стойността на функцията $f(x_1, \dots, x_n)$:

$$f(x_1, \dots, x_n) = \begin{cases} 1, & \text{ако } x_1 \leq \dots \leq x_n \\ \sum_{i=1}^{n-1} e^{x_i + x_{i+1}}, & \text{в останалите случаи} \end{cases}$$

За всяка стойност y на променливата x , стойността e^y се пресмята с точност 10^{-6} и методът извежда y , `MyMath.exp(y)` и `Math.exp(y)` с цел проверка, че e^y се определя правилно.

```
package Library;

public class MyMath {
    public static double E = 1E-12;

    public static double sqrt(double arg) {
        if (arg < 0)
            return Double.NaN;
        double y0, y1 = 1;
        do {
            y0 = y1;
            y1 = (arg / y0 + y0) / 2.0;
        } while (Math.abs(y0 - y1) >= E);
        return y1;
    }

    public static double PI() {
        int k = 0;
        double four = 4.0;
        double term = four;
        double sum = term;
        while (Math.abs(term) >= E) {
            k++;
            four = -four;
            term = four / (2 * k + 1);
            sum += term;
        }
        return sum;
    }

    public static double exp(double arg) {
        double sum = 1, term = 1;
        int number = 0;
        while (Math.abs(term) >= E) {
            number++;
            term = term * arg / number;
            sum += term;
        }
        return sum;
    }

    public static double E() {
        System.out.println("Not implemented!");
        return Double.NaN;
    }

    public static double sin(double arg) {
        System.out.println("Not implemented!");
        return Double.NaN;
    }

    public static double cos(double arg) {
```

```

        System.out.println("Not implemented!");
        return Double.NaN;
    }

    public static double arctg(double arg) {
        System.out.println("Not implemented!");
        return Double.NaN;
    }
}

```

Тестване:

```

package Tests;

import java.util.Scanner;
import Library.MyMath;

public class TestMyMath {
    public static Scanner sc = new Scanner(System.in);

    public static void testSqrt() {
        System.out.print("Enter n>0: ");
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            System.out.print("      Enter x>0: ");
            double x = sc.nextDouble();
            System.out.println("      sqrt(x) = " + MyMath.sqrt(x));
            System.out.println("      Math.sqrt(x) = " + Math.sqrt(x));
        }
    }

    public static void testPI() {
        System.out.println("Start...");
        System.out.print("Enter 0<epsilon<1: ");
        double epsilon = sc.nextDouble();
        while (0 < epsilon && epsilon < 1) {
            MyMath.E = epsilon;
            double PI = MyMath.PI();
            System.out.println("      PI = " + PI);
            System.out.println("      Math.PI = " + Math.PI);
            System.out.println(Math.abs(PI - Math.PI) < MyMath.E ? "Yes" : "No");
            System.out.print("Enter 0<epsilon<1: ");
            epsilon = sc.nextDouble();
        }
        System.out.println("End...");
    }

    public static void testExp() {
        System.out.print("Enter an integer n>=2: ");
        int n = sc.nextInt();
        if (n < 2)
            System.out.println("Incorrect n = " + n + ". Try again!");
        else {
            double result = 0;
            boolean f = true;
            MyMath.E = 1.0E-6;
            System.out.print("      Enter x1 = ");
            double x1 = sc.nextDouble();
            for (int i = 2; i <= n; i++) {
                System.out.print("      Enter x" + i + " = ");
                double x2 = sc.nextDouble();
                f = f && (x1 <= x2);
                double y = x1 + x2;
                double sum = MyMath.exp(y);
                System.out.println("      y = " + y);
            }
        }
    }
}

```

```

        System.out.println("      exp(y) = " + sum);
        System.out.println("      Math.exp(y) = " + Math.exp(y));
        result += sum;
        x1 = x2;
    }
    if (f)
        result = 1;
    System.out.println();
    System.out.println("      Result = " + result);
}
}

public static void testE() {
    System.out.println("testE is not implemented!");
}

public static void testSin() {
    System.out.println("testSin is not implemented!");
}

public static void testCos() {
    System.out.println("testCos is not implemented!");
}

public static void testArctg() {
    System.out.println("testArctg is not implemented!");
}

public static void main(String[] args) {
    System.out.println("1. testE");
    System.out.println("2. testPI");
    System.out.println("3. testSqrt");
    System.out.println("4. testExp");
    System.out.println("5. testSin");
    System.out.println("6. testCos");
    System.out.println("7. testArctg");
    System.out.println("8. End of testing");
    int n;
    do {
        System.out.print("Enter the number of test method: ");
        n = sc.nextInt();
        switch (n) {
            case 1:
                testE();
                break;
            case 2:
                testPI();
                break;
            case 3:
                testSqrt();
                break;
            case 4:
                testExp();
                break;
            case 5:
                testSin();
                break;
            case 6:
                testCos();
                break;
            case 7:
                testArctg();
                break;
        }
    } while (1 <= n && n <= 7);
    System.out.println("Done!");
}

```

```
}
```

Тестване на метода PI:

Първо изпълнение:

```
1. testE
2. testPI
3. testSqrt
4. testExp
5. testSin
6. testCos
7. testArctg
8. End of testing
Enter the number of test method: 2
Start...
Enter 0<epsilon<1: 1E-1
    PI = 3.189184782277596
    Math.PI = 3.141592653589793
Yes
Enter 0<epsilon<1: 1E-2
    PI = 3.1465677471829556
    Math.PI = 3.141592653589793
Yes
Enter 0<epsilon<1: -1
End...
Enter the number of test method: 8
Done!
```

Второ изпълнение:

```
1. testE
2. testPI
3. testSqrt
4. testExp
5. testSin
6. testCos
7. testArctg
8. End of testing
Enter the number of test method: 2
Start...
Enter 0<epsilon<1: 1E-6
    PI = 3.1415931535894743
    Math.PI = 3.141592653589793
Yes
Enter 0<epsilon<1: 1E-8
    PI = 3.1415926585894076
    Math.PI = 3.141592653589793
Yes
Enter 0<epsilon<1: -1
End...
Enter the number of test method: 8
Done!
```

Тестване на метода exp:

Първо изпълнение:

```
1. testE
2. testPI
3. testSqrt
4. testExp
5. testSin
6. testCos
7. testArctg
8. End of testing
Enter the number of test method: 5
testSin is not implemented!
Enter the number of test method: 4
Enter an integer n>=2: 3
    Enter x1=1
    Enter x2=2
        y = 3.0
        exp(y) = 20.08553685145113
    Math.exp(y) = 20.085536923187668
Enter x3=3
    y = 5.0
    exp(y) = 148.41315898276954
Math.exp(y) = 148.4131591025766

    Result = 1.0
Enter the number of test method: 8
Done!
```

Второ изпълнение:

```
1. testE
2. testPI
3. testSqrt
4. testExp
5. testSin
6. testCos
7. testArctg
8. End of testing
Enter the number of test method: 4
Enter an integer n>=2: 3
    Enter x1=3
    Enter x2=1
        y = 4.0
        exp(y) = 54.598149928148814
    Math.exp(y) = 54.598150033144236
Enter x3=2
    y = 3.0
    exp(y) = 20.08553685145113
    Math.exp(y) = 20.085536923187668

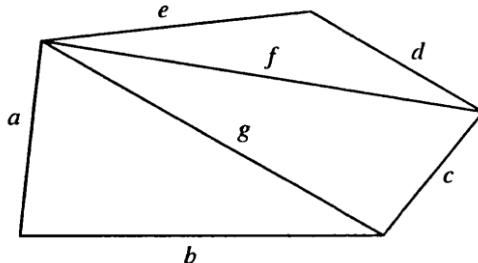
    Result = 74.68368677959995
Enter the number of test method: 8
Done!
```



Задачи за упражнение

Задача: Да се състави програма, която за дадени цели числа a, b, c определя най-големия им общ делител НОД(a,b,c) = НОД(НОД(a,b),c). Програмата да съдържа метод за определяне на най-големия общ делител на две цели положителни числа по алгоритъма на Евклид.

Задача: Да се състави програма, която за дадени реални числа a, b, c, d, e, f, g определя лицето на дадения петоъгълник. Програмата да съдържа метод за пресмятане на лице на триъгълник по трите му страни.



Задача: Да се състави програма, която за дадено n пресмята:

$$\sqrt{3 + \sqrt{6 + \dots + \sqrt{3(n-1) + \sqrt{3n}}}}$$

Програмата да съдържа метод sqrt(x) за пресмятане на корен квадратен от реално число x.

Задача: Да се състави програма, която за дадено n пресмята:

$$\frac{1}{\sin 1} + \frac{1}{\sin 1 + \sin 2} + \dots + \frac{1}{\sin 1 + \dots + \sin n}$$

Програмата да съдържа метод за пресмятане на sin(x) от реално число x – ъгъл в радиани.

Задача: Да се състави програма, която за дадено n пресмята:

$$\frac{\cos 1}{\sin 1} + \frac{\cos 1 + \cos 2}{\sin 1 + \sin 2} + \dots + \frac{\cos 1 + \dots + \cos n}{\sin 1 + \dots + \sin 2n}$$

Програмата да съдържа методи за пресмятане на sin(x) и cos(x) от реално число x – ъгъл в радиани.

Задача: Да се реализират:

1. Клас **Convertor** от пакета **Library** за преобразуване от една мерна единица в друга съгласно дадената таблица. За всяко n класът да предоставя по два метода: единият има параметър за лявата мерна единица и го преобразува в дясната мерна единица, а другият метод има параметър за дясната мерна единица и го преобразува в лявата мерна единица.

n	
1	1 унция (за течности) = 29.586 милилитра
2	1 галон = 3.785 литра
3	1 унция = 28.3495 грама
4	1 паунд = 453.6 грама
5	1 инч = 2.54 сантиметра
6	1 фут = 30.5 сантиметра
7	1 миля = 1.609 километра
8	1 градус (за ъгли) = $\pi/180$ радиана
9	C градуса по Целзий = $5/9(F-32)$, F градуса по Фаренхайт

2. Клас **TestConvertor** от пакета **Tests** за тестване на методите от класа **Convertor**



Тип масив

Масивът съдържа 0 или повече променливи без имена, наречени *компоненти*, като достъпът до тях е чрез *индекс*. Ако масив има n компоненти, казваме че той има дължина n и достъпът до неговите компонентите е чрез целочислени индекси от 0 до n-1, включително. Всички компоненти на масив са от един и същ тип – *тип на компонентите*. Ако той е T, то типът на самия масив е T[]. Типът T също може да е тип масив и т.н., като типът на компонентите, който не е масив се нарича *тип на елементите на масива*.

Тип масив се записва като тип на елементите на масива, след който има една или повече двойки скоби []. Такъв тип може да бъде всеки примитивен или съставен тип. Типовете масиви се използват в операцията за преобразуване на типа и при декларации.

Стойностите на тип масив са адреси на обекти от този тип и над тях могат да се прилагат *операциите за съставни типове*:

- Достъп до поле
- Извикване на метод
- Операция за преобразуване на типа
- Операция за конкатениране на низове +
- Операция instanceof
- Операции за сравнение за равно == и различно !=
- Условна операция ?:

1. Деклариране на променлива от тип масив, създаване и инициализиране на масив

В декларацията на променлива може да се зададе тип масив. Такава променлива може да съдържа адрес на масив (обект) от посочения тип. Масив се *създава* и *инициализира* съгласно спецификацията на езика Java.

Декларација:

```
int a[], b[][]; или int[] a, b[]; или int[] a; int[][] b; a [ ] ? b [ ] ?
```

Създаване:

```
a = new int[10]; a [ ] → Поле за запис на 10 цели,  
инициализирано с нули
```

```
b = new int[10][20] b [ ] → [ ] → Поле за запис на 20 цели,  
инициализирано с нули  
[ ] → [ ] → Поле за запис на 20 цели,  
инициализирано с нули
```

Създаване и инициализиране:

```
int[] a = {1, 2, 3, 4, 5}; a [ ] → [1 | 2 | 3 | 4 | 5]
```

Чрез оператор за присвояване на променлива от тип масив може да се присвои стойността на променлива от същия тип масив. Ако типът не е същият, по време на транслация се получава грешка.

Операциите за сравнение за равно `==` и различно `!=` може да се прилагат над променливи от един и същ тип масив. В противен случай по време на транслация се получава грешка.

2. Достъп до компонентите на масив

Операцията за достъп до компонента на масив от тип `T[]` има вида

Име_на_масив [Аритметичен_израз]

Определеният по време на изпълнение тип на израза трябва да е `int`. Ако стойността на променливата масив е `null`, изпълнението приключва с активиране на изключението `NullPointerException`. В противен случай се изчислява изразът и ако неговата стойност е допустим индекс на компонента, то резултатът на операцията е компонентата с този индекс. В противен случай изпълнението приключва с активиране на изключението `ArrayIndexOutOfBoundsException`.

3. Локални масиви. Масиви на клас. Параметри-масиви и методи, които връщат масив

Локални масиви и масиви на клас се декларират по същия начин както и съответните променливи. Операцията за достъп до масив на клас има вида

Име_на_клас.Име_на_масив

Метод може да има параметър-масив, както и да връща резултат от тип масив. Типът на фактическия параметър трябва да съвпада с типа на формалния параметър.

Задача: Да се състави програма, която чете `n` числа и ги извежда в ред обратен на въвеждането.

Реализация: Програма **Example4_1**:

```
package Arrays;

import java.util.*;

public class Example4_1 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter n>0: ");
        int n = sc.nextInt();
        double numbers[] = new double[n];

        // Input:
        System.out.println("Input:");
        for (int i = 1; i <= n; i++) {
            System.out.print("number[" + i + "]: ");
            numbers[i - 1] = sc.nextDouble();
        }

        // Output:
        System.out.println("Output:");
        for (int i = n - 1; i >= 0; i--) {
            System.out.println("number[" + (i + 1) + "] = " + numbers[i]);
        }
    }
}
```

Изпълнение на програмата:

Първо изпълнение: Второ изпълнение: Трето изпълнение:

Enter n>0: 3 Input: number[1]: 1 number[2]: 2 number[3]: 3 Output: number[3] = 3.0 number[2] = 2.0 number[1] = 1.0	Enter n>0: 0 Input: Output:	Enter n>0: -3 Exception in thread "main" java.lang.NegativeArraySizeException at Example4_1.main(Example4_1.java:8)
--	-----------------------------------	---

Предложеният вариант на програма се състои от клас **Example4_1**, съдържащ метод main, в който се декларира и използва локален масив numbers. Освен този вариант, може да са реализират още два варианта:

Вариант 1: Клас **Example4_1_1** със следната спецификация:

```
public class Example4_1_1 {
    public static double numbers[];
    //Декларация на масив на клас numbers
    public static void read() {...}
    //Създава масива numbers, чете числата и ги съхранява в него
    public static void print() {...}
    //Извежда числата от масива numbers в ред обратен на въвеждането им
    public static void main(String[] args) {...}
    //Извиква двета метода
}
```

Реализация:

```
package Arrays;

import java.util.Scanner;

public class Example4_1_1 {
    // Data
    public static double numbers[];

    // Methods
    public static void read() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter n>0: ");
        int n = sc.nextInt();
        numbers = new double[n];
        // Input:
        System.out.println("      Input:");
        for (int i = 1; i <= n; i++) {
            System.out.print("Enter numbers[" + i + "]:");
            numbers[i - 1] = sc.nextDouble();
        }
    }

    public static void print() {
        System.out.println("      Output:");
        for (int i = numbers.length - 1; i >= 0; i--)
            System.out.println("numbers[" + (i + 1) + "]=" + numbers[i]);
    }

    public static void main(String[] args) {
        read();
        print();
    }
}
```

Вариант 2: Клас Example4_1_2 със следната спецификация:

```
public class Example4_1_2 {
    public static double[] read() {...}
    //Декларира и създава локален масив от реални числа. Чете числата и ги съхраняват в
    //него. Връща като резултат получния масив
    public static void print(double[] numbers) {...}
    //Извежда числата от параметъра-масив numbers в ред обратен на въвеждането им
    public static void main(String[] args) {...}
    //Декларира локален масив и го създава и инициализира, използвайки метода read.
    //Извежда числата, използвайки метода print
}
```

Реализация:

```
package Arrays;

import java.util.Scanner;

public class Example4_1_2 {
    // Methods
    public static double[] read() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter n>0: ");
        int n = sc.nextInt();
        double numbers[] = new double[n];
        // Input:
        System.out.println("      Input:");
        for (int i = 1; i <= n; i++) {
            System.out.print("Enter numbers[" + i + "]:");
            numbers[i - 1] = sc.nextDouble();
        }
        return numbers;
    }

    public static void print(double[] numbers) {
        System.out.println("      Output:");
        for (int i = numbers.length - 1; i >= 0; i--)
            System.out.println("numbers[" + (i + 1) + "]=" + numbers[i]);
    }

    public static void main(String[] args) {
        double numbers[] = read();
        print(numbers);
    }
}
```

Използване на масиви

Следващите програми илюстрират използването на масиви за:

- Съхранение на еднотипни обекти, подлежащи на обработка, например числа, полиноми, матрици
- Съхранение на информация за обектите, подлежащи на обработка, например за последователност от цели неотрицателни десетични числа i -тият елемент на масив съдържа true, ако числото i се среща в последователността и false в противен случай
- Представяне на обект, подлежащ на обработка, например полином, матрица

Задача: Да се състави програма, която определя най-късия интервал, който съдържа n дадени числа a_1, \dots, a_n .

Решение: Най-късият интервал е $[min(a_1, \dots, a_n); max(a_1, \dots, a_n)]$.

Спецификация на програмата:

```
public class Example4_2 {
    public static double[] numbers;
    //Декларация на масив на клас numbers
    public static void read() {...}
    //Създава масива numbers, чете числата и ги съхранява в него
    public static double min() {...}
    //Определя минимума на елементите на масива numbers
    public static double max() {...}
    //Определя максимума на елементите на масива numbers
    public static void solution() {...}
    //Извиква метода read и извежда интервала, използвайки методите min и max
    public static void main(String[] args) {...}
    //Извиква метода solution
}
```

Реализация: Програма Example4_2:

```
package Arrays;

import java.util.Scanner;

public class Example4_2 {
    // Data
    public static double[] numbers;

    // Methods
    public static void read() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter n>0: ");
        int n = sc.nextInt();
        numbers = new double[n];
        for (int i = 0; i < numbers.length; i++) {
            System.out.print("Enter a real number: ");
            numbers[i] = sc.nextDouble();
        }
    }

    public static double min() {
        double result = Double.POSITIVE_INFINITY;
        for (int i = 0; i < numbers.length; i++)
            if (result > numbers[i])
                result = numbers[i];
        return result;
    }

    public static double max() {
        double result = Double.NEGATIVE_INFINITY;
        for (int i = 0; i < numbers.length; i++)
            if (result < numbers[i])
                result = numbers[i];
        return result;
    }

    public static void solution() {
        read();
        System.out.println("[" + min() + ";" + max() + "]");
    }

    public static void main(String[] args) {
        solution();
    }
}
```

Задача: Да се състави програма, която за п дадени цели числа определя броя на едноцифрените числа, броя на двуцифрените числа, ..., броя на дванадесетцифрените числа.

Спецификация на програмата:

```
public class Example4_3 {
    public static int length(int arg) {...}
    //Определя броя на цифрите в записа на цялото число arg
    public static void fill(int[] numbers,int value) {...}
    //Присвоява на всеки елемент на параметъра-масив numbers стойност value
    public static void print(int[] numbers) {...}
    //Извежда числата от параметъра-масив numbers
    public static void solution() {...}
    //Декларира и създава локален масив. Инициализира го с нули, използвайки метода fill
    //Обработва входната последователност от числа. Извежда елементите на масива,
    //използвайки метода print
    public static void main(String[] args) {...}
    //Извиква метода solution
}
```

Реализация: Програма Example4_3:

```
package Arrays;

import java.util.Scanner;

public class Example4_3 {
    // Methods
    public static int length(int arg) {
        arg = Math.abs(arg);
        int count = 0;
        do {
            count++;
            arg /= 10;
        } while (arg != 0);
        return count;
    }

    public static void fill(int[] numbers, int value) {
        for (int i = 0; i < numbers.length; i++)
            numbers[i] = value;
    }

    public static void print(int[] numbers) {
        for (int i = 0; i < numbers.length; i++)
            System.out.println("Count of integers of length " + (i + 1)
                + " is " + numbers[i]);
    }

    public static void solution() {
        int[] counts = new int[12];
        fill(counts, 0);
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter n>0: ");
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            System.out.print("Enter an integer: ");
            int number = sc.nextInt();
            counts[length(number) - 1]++;
        }
        print(counts);
    }
}
```

```

public static void main(String[] args) {
    solution();
}
}

```

Задача: Да се състави програма, която за дадено $n > 0$ създава следната таблица и я извежда.

НОД(1, 1)	НОД(1, 2)	...	НОД(1, n)
НОД(2, 1)	НОД(2, 2)	...	НОД(2, n)
...
НОД(n, 1)	НОД(n, 2)	...	НОД(n, n)

Спецификация на програмата:

```

public class Example4_4 {
    public static int gcd(int a,int b) {...}
    //Определя най-големия общ делител на целите числа a и b
    public static int[][] create(int n) {...}
    //Декларира и създава локален масив. Инициализира го, използвайки метода gcd.
    public static void print(int[][] numbers) {...}
    //Извежда числата от параметъра-масив numbers
    public static void solution() {...}
    //Декларира и създава локален масив. Инициализира го, използвайки метода create.
    //Извежда елементите на масива, използвайки метода print
    public static void main(String[] args) {...}
    //Извиква метода solution
}

```

Реализация: Програма Example4_4:

```

package Arrays;

import java.util.Scanner;

public class Example4_4 {
    // Methods
    public static int gcd(int a, int b) {
        int p = a % b;
        while (p != 0) {
            a = b;
            b = p;
            p = a % b;
        }
        return b;
    }

    public static int[][] create(int n) {
        int[][] result = new int[n][n];
        for (int i = 0; i < result.length; i++) {
            for (int j = 0; j < result[0].length; j++)
                result[i][j] = gcd(i + 1, j + 1);
        }
        return result;
    }

    public static void print(int[][] numbers) {
        for (int i = 0; i < numbers.length; i++) {
            for (int j = 0; j < numbers[0].length; j++)
                System.out.print(numbers[i][j] + " ");
            System.out.println();
        }
    }
}

```

```

public static void solution() {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter n>0: ");
    int n = sc.nextInt();
    int[][] table = create(n);
    print(table);
}

public static void main(String[] args) {
    solution();
}
}

```

Задача: Да се състави програма, която проверява дали дадена последователност от n числа е симетрична относно средата си. Например, последователностите 10, 89, 41, 41, 89, 10 и 9, 11, 5, 11, 9 са такива, а 12, 234, 45, 18, 12 - не.

Спецификация на програмата:

```

public class Example4_5 {
    public static double[] numbers;
    //Декларация на масив на клас numbers
    public static void read() {...}
    //Създава масива numbers, чете числата и ги съхранява в него
    public static boolean isSymmetric() {...}
    //Проверява дали масива numbers съдържа симетрична последователност
    public static void solution() {...}
    //Извиква метода read. Извежда подходящо съобщение за резултата от проверката,
    //използвайки метода isSymmetric
    public static void main(String[] args) {...}
    //Извиква метода solution
}

```

Реализация: Програма Example4_5:

```

package Arrays;

import java.util.Scanner;

public class Example4_5 {
    // Data
    public static double[] numbers;

    // Methods
    public static void read() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter n>0: ");
        int n = sc.nextInt();
        numbers = new double[n];
        for (int i = 0; i < numbers.length; i++) {
            System.out.print("Enter a real number: ");
            numbers[i] = sc.nextDouble();
        }
    }

    public static boolean isSymmetric() {
        boolean flag = true;
        int middle = numbers.length / 2;
        for (int i = 0; i < middle; i++)
            flag = flag && (numbers[i] == numbers[numbers.length - 1 - i]);
        return flag;
    }
}

```

```

public static void solution() {
    read();
    System.out.println(isSymmetric() ? "Yes" : "No");
}

public static void main(String[] args) {
    solution();
}
}

```

Задача: Да се състави програма, която чете последователност от цели положителни числа, завършваща с -1 и ги записва (без числото -1) в масив с дължина $n > 0$ по следния начин:

- Позицията p , където следва да се запише поредното число a , се определя по формулата $p = a \% n$

- Ако позицията p е заета, то се търси първото свободно място за запис, като се обхождат позициите: $p+1, p+2, \dots, n-2, n-1, 0, 1, \dots, p-1$

Ако бъде намерена свободна позиция, числото a се записва в нея. Ако няма свободно място за запис в масива, въвеждането на числата се прекратява и изпълнението на програмата завършва след извеждане на подходящо съобщение.

Реализация: Програма Example4_6:

```

package Arrays;

import java.util.Scanner;

public class Example4_6 {

    public static void main(String[] args) {
        int a, p, q;
        Scanner sc = new Scanner(System.in);
        System.out.print("      Enter n>0: ");
        int n = sc.nextInt();
        int tab[] = new int[n];
        System.out.println("      Enter positive integers. End=-1.");
        for (int i = 0; i < n; i++)
            tab[i] = 0;
        while ((a = sc.nextInt()) != -1) {
            p = a % n;
            if (tab[p] == 0)
                tab[p] = a;
            else {
                q = p;
                p = (p + 1) % n;
                while ((p != q) && (tab[p] != 0))
                    p = (p + 1) % n;
                if (p == q) {
                    System.out.println("No space!");
                    break;
                } else
                    tab[p] = a;
            }
        }
        for (int i = 0; i < n; i++)
            System.out.println("Tab[" + i + "]=" + tab[i]);
    }
}

```

Изпълнение на програмата:

Първо изпълнение:

```
Enter n>0: 5
Enter positive
integers.End=-1.
10 3 23 1 -1
Tab[0]=10
Tab[1]=1
Tab[2]=0
Tab[3]=3
Tab[4]=23
```

Второ изпълнение:

```
Enter n>0: 8
Enter positive
integers.End=-1.
13 11 21 7 5 6 5 5 9 10
-1
No space!
Tab[0]=5
Tab[1]=6
Tab[2]=5
Tab[3]=11
Tab[4]=5
Tab[5]=13
Tab[6]=21
Tab[7]=7
```

Трето изпълнение:

```
Enter n>0: 6
Enter positive
integers.End=-1.
11 10 9 8 7 6 -1
Tab[0]=6
Tab[1]=7
Tab[2]=8
Tab[3]=9
Tab[4]=10
Tab[5]=11
```

Задача: Да се състави програма, която пресмята стойността на даден полином на една променлива с реални коефициенти $P(x) = a_nx^n + a_{n-1}x^{n-1} + \dots + a_0$ за дадена стойност на променливата x по правилото на Хорнер $P(x) = (((a_nx + a_{n-1})x + a_{n-2})x + \dots) + a_0$.

Спецификация на програмата:

```
public class Example4_7 {
    public static double[] read() {...}
    //Създава масив double[] result=new double[n+1]. Чете коефициентите ai, i∈[0;n] на полином от
    //степен n и ги съхранява в него, като result[i]=ai
    public static double eval(double[] arg,double value) {...}
    //Пресмята стойността на полинома arg за стойност на променливата value по правилото на Хорнер
    public static void solution() {...}
    //Извиква метода read и определя стойността на полинома, използвайки метода eval
    public static void main(String[] args) {...}
    //Извиква метода solution
}
```

Реализация: Програма Example4_7:

```
package Arrays;

import java.util.Scanner;

public class Example4_7 {
    // Methods
    public static double[] read() {
        Scanner sc = new Scanner(System.in);
        System.out.print("      Enter n≥0: ");
        int n = sc.nextInt();
        double[] result = new double[n + 1];
        for (int i = n; i ≥ 0; i--) {
            System.out.print("Coeff[" + i + "] = ");
            result[i] = sc.nextDouble();
        }
        return result;
    }

    public static double eval(double[] arg, double value) {
        double result = 0;
        for (int i = arg.length - 1; i ≥ 0; i--)
            result = result * value + arg[i];
        return result;
    }

    public static void solution() {
        System.out.println("      Enter polynom P(x): ");
    }
}
```

```

double[] P = read();
Scanner sc = new Scanner(System.in);
System.out.print("      Enter value: ");
double value = sc.nextDouble();
System.out.println("P(" + value + ") = " + eval(P, value));

}

public static void main(String[] args) {
    solution();
}
}

```

Задача: Да се състави програма, която чете чисрова матрица $A(n \times n)$ и проверява дали тя е ортогонална, т.е. $AA^T = E$, където A^T е транспонираната матрица на A , а E е единична матрица.

Спецификация на програмата:

```

public class Example4_8 {
    public static double[][] E(int n) {...}
    //Създава единична матрица nxn
    public static double[][] read() {...}
    //Създава масив double[][] result=new double[n][m]. Чете елементите  $a_{ij}$ ,  $i \in [1;n]$  и  $j \in [1;m]$ , на матрица
    //nxm и ги съхранява в него, като result[i-1][j-1]= $a_{ij}$ , result.length=n, result[0].length=m
    public static void print(double[][] arg) {...}
    //Извежда матрицата arg
    public static double[][] mult(double[][] arg1, double[][] arg2) {...}
    //Получава произведението на матриците arg1 и arg2: arg1*arg2
    public static double[][] transpose(double[][] arg) {...}
    //Получава транспонираната матрица на матрицата arg
    public static boolean equal(double[][] arg1, double[][] arg2) {...}
    //Проверява дали матриците arg1 и arg2 съвпадат
}

```

Реализация: Програма Example4_8:

```

package Arrays;

import java.util.Scanner;

public class Example4_8 {
    public static double[][] E(int n) {
        double[][] result = new double[n][n];
        for (int i = 0; i < n; i++)
            result[i][i] = 1;
        return result;
    }

    public static double[][] read() {
        Scanner sc = new Scanner(System.in);
        System.out.print("      Enter n>0: ");
        int n = sc.nextInt();
        System.out.print("      Enter m>0: ");
        int m = sc.nextInt();
        double[][] result = new double[n][m];
        System.out.println("      Enter elements: ");
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
                result[i - 1][j - 1] = sc.nextDouble();
        return result;
    }

    public static void print(double[][] arg) {

```

```

for (int i = 0; i < arg.length; i++) {
    System.out.println();
    for (int j = 0; j < arg[0].length; j++)
        System.out.print(arg[i][j] + "   ");
}
}

public static double[][] mult(double[][] arg1, double[][] arg2) {
    if (arg1[0].length != arg2.length) {
        System.out.println("Unsupported Operation: multiplication!");
        return null;
    }
    double[][] result = new double[arg1.length][arg2[0].length];
    for (int i = 0; i < result.length; i++) {
        for (int j = 0; j < result[0].length; j++) {
            result[i][j] = 0;
            for (int k = 0; k < arg1[0].length; k++)
                result[i][j] += (arg1[i][k] * arg2[k][j]);
        }
    }
    return result;
}

public static double[][] transpose(double[][] arg) {
    if (arg.length != arg[0].length) {
        System.out.println("Unsupported Operation: transpose!");
        return null;
    }
    double[][] result = new double[arg.length][arg[0].length];
    for (int i = 0; i < result.length; i++) {
        for (int j = 0; j < result[0].length; j++)
            result[i][j] = arg[j][i];
    }
    return result;
}

public static boolean equal(double[][] arg1, double[][] arg2) {
    if ((arg1.length != arg2.length) || (arg1[0].length != arg2[0].length))
        return false;
    for (int i = 0; i < arg1.length; i++) {
        for (int j = 0; j < arg1[0].length; j++)
            if (arg1[i][j] != arg2[i][j])
                return false;
    }
    return true;
}

public static void main(String[] args) {
    double[][] A = read();
    double[][] T = transpose(A);
    print(T);
    T = mult(A, T);
    print(T);
    double[][] E = E(A.length);
    System.out.println(equal(T, E));
}
}

```

Задача: Да се състави програма, която получава сортиран масив от различните елементи на даден масив от числа.

Решение: Ще формулираме два основни алгоритъма за решаване на задачата:

Алгоритъм 1: Създаване на масив от различните елементи на дадения масива. Сортиране във възходящ ред на получния масив.

Алгоритъм 2: Създаване на масив, сортиран във възходящ ред, от различните елементи на дадения масив, като се използва метода за включване в сортиран масив.

Алгоритъм 1 Solution: Даден е числов масив A. Като резултат се получава сортиран във възходящ ред масив B от различните елементи на A.

1. $B = \text{differentElements}(A)$
2. $\text{bubbleSort}(B)$

Нека $\text{array} = (e_1, \dots, e_n)$ е числов масив. Функцията $\text{differentElements(array)}$ създава масив от различните елементи на масива array. Процедурата bubbleSort(array) сортира (пренарежда) по метода на мехурчето елементите на масива array, така че $e_{i_1} \leq \dots \leq e_{i_n}$.

Алгоритъм DifferentElements: Даден е числов масив $\text{array} = (a_1, \dots, a_n)$. Като резултат се получава масив $\text{result} = (b_1, \dots, b_k)$, $k \leq n$, от различните елементи на array.

1. $k = 0$
2. $i = 1$
3. Ако $i \leq n$, преход на т.4. Иначе – край на алгоритъма
4. $\text{index} = \text{linearSearch}(a_i, B)$
5. Ако $\text{index} = -1$, преход на т.6. Иначе – преход на т.8
6. $k = k + 1$
7. $b_k = a_i$
8. $i = i + 1$
9. Преход на т. 3

Нека $\text{array} = (e_1, \dots, e_n)$ е числов масив. Функцията $\text{linearSearch(value, array)}$ проверява по метода на последователното търсене дали value съвпада с някой от елементите на масива e_1, \dots, e_n . Ако не съвпада, връща -1 . В противен случай връща минималния индекс на елемент, съвпадащ с value.

Алгоритъм LinearSearch: Даден е числов масив $\text{array} = (e_1, \dots, e_n)$ и число value. Като резултат се получава стойността на променливата result: минималния индекс на елемент на масива array, съвпадащ с value или -1 , ако няма такъв.

1. $\text{result} = -1$
2. $i = 1$
3. Ако $i \leq n$, преход на т.4. Иначе – преход на т.9
4. Ако $e_i = \text{value}$, преход на т.5. Иначе – преход на т.7
5. $\text{result} = i$
6. Преход на т.9
7. $i = i + 1$
8. Преход на т.3
9. Край на алгоритъма

Алгоритъм BubbleSort: Даден е числов масив $\text{array} = (e_1, \dots, e_n)$. Като резултат се получава масива array, сортиран във възходящ ред.

1. $\text{sorted} = \text{true}$
2. $i = 1$
3. Ако $i \leq n-1$, преход на т.4. Иначе – преход на т.11
4. Ако $e_i > e_{i+1}$, преход на т.5. Иначе – преход на т.9
5. $\text{temp} = e_i$
6. $e_i = e_{i+1}$
7. $e_{i+1} = \text{temp}$
8. $\text{sorted} = \text{false}$

9. $i = i + 1$
10. Преход на т.3
11. Ако sorted = false, преход на т.1. Иначе – край на алгоритъма

Реализация: Програма **Example4_9**:

```

package Arrays;

import java.util.Scanner;

public class Example4_9 {
    // Methods
    public static double[] differentElements(double[] array) {
        double[] B = new double[array.length];

        int k = -1;
        for (int i = 0; i < array.length; i++) {
            int index = linearSearch(array[i], B, 0, k);
            if (index == -1)
                B[++k] = array[i];
        }

        double[] result = new double[k + 1];
        for (int i = 0; i < result.length; i++)
            result[i] = B[i];
        return result;
    }

    public static int linearSearch(double value, double[] array, int left,
        int right) {
        int result = -1;
        for (int i = left; i <= right; i++)
            if (array[i] == value) {
                result = i;
                break;
            }
        return result;
    }

    public static int linearSearch(double value, double[] array) {
        return linearSearch(value, array, 0, array.length - 1);
    }

    public static void bubbleSort(double[] array) {
        boolean sorted;
        do {
            sorted = true;
            for (int i = 0; i < array.length - 1; i++)
                if (array[i] > array[i + 1]) {
                    double temp = array[i];
                    array[i] = array[i + 1];
                    array[i + 1] = temp;
                    sorted = false;
                }
        } while (!sorted);
    }

    public static double[] read() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter n>0: ");
        int n = sc.nextInt();
        double[] numbers = new double[n];
        for (int i = 0; i < numbers.length; i++) {
            System.out.print("Enter a real number: ");
            numbers[i] = sc.nextDouble();
        }
    }
}

```

```

    }
    return numbers;
}

public static void print(double[] numbers) {
    for (int i = 0; i < numbers.length; i++)
        System.out.println(numbers[i]);
}

public static void solution() {
    double[] A = read();
    System.out.println("      Different elements:");
    double[] B = differentElements(A);
    print(B);
    System.out.println("      Sorted different elements:");
    bubbleSort(B);
    print(B);
}

public static void main(String[] args) {
    solution();
}
}

```

Изпълнение на програмата:

Първо изпълнение:

```

Enter the number of
elements: 5
Enter the elements:
Enter element #1: 77
Enter element #2: 22
Enter element #3: 44
Enter element #4: 22
Enter element #5: 77
      Different elements:
The number of elements:
3
Element #1: 22.0
Element #2: 44.0
Element #3: 77.0

```

Второ изпълнение:

```

Enter the number of
elements: 5
Enter the elements:
Enter element #1: 5
Enter element #2: 4
Enter element #3: 3
Enter element #4: 2
Enter element #5: 1
      Different elements:
The number of elements:
5
Element #1: 1.0
Element #2: 2.0
Element #3: 3.0
Element #4: 4.0
Element #5: 5.0

```

Трето изпълнение:

```

Enter the number of
elements: 5
Enter the elements:
Enter element #1: 11
Enter element #2: 11
Enter element #3: 11
Enter element #4: 11
Enter element #5: 11
      Different elements:
The number of elements:
1
Element #1: 11.0

```

Задача: Да се състави програма, която проверява дали елементите на даден числов масив numbers с дължина n съдържа всяко от числата от 1 до n.

Решение: Ще формулираме два типа алгоритми за решаване на задачата.

Тип 1: Първият тип алгоритми се състои в използването на масива numbers:

Алгоритъм 1: Проверка дали всяко от числата от 1 до n се среща в масива numbers.

Алгоритъм 2: Сортиране на масива numbers и проверка дали за елементите му е изпълнено numbers[0]=1, numbers[1]=2,..., numbers[n-1]=n.

Тип 2: Вторият тип алгоритми се основава на създаването на масив info с n елемента, съдържащ информация за елементите на масива numbers:

Алгоритъм 1: Създаване на масив info с n елемента от числов тип, като за всяко число $a \in [1;n]$ елементът info[a-1] съдържа брой срещания на числото a в масива numbers, както следва: инициализация на всеки елемент на масива info с 0, след което масивът numbers се

обхожда и за всеки негов елемент $1 \leq b \leq n$ елементът $\text{info}[b-1]$ се увеличава с 1. Проверка дали за елементите на масива info е изпълнено $\text{info}[0] = \dots = \text{info}[n-1] = 1$.

Алгоритъм 2: Създаване на масив info с n елемента от тип `boolean`, като за всяко число $a \in [1; n]$ елементът $\text{info}[a-1]$ съдържа стойност `true`, ако числото a се среща в масива numbers и `false` – в противен случай, както следва: инициализация на всеки елемент на масива info с `false`, след което масивът numbers се обхожда и за всеки негов елемент $1 \leq b \leq n$ елементът $\text{info}[b-1]$ се променя на `true`. Проверка дали за елементите на масива info е изпълнено $\text{info}[0] = \dots = \text{info}[n-1] = \text{true}$.

Задача: Да се съставят и тестват програмни средства, които за даден масив от числа определят различните числа в масива и колко пъти всяко се среща в него. Например, за масива $A = (32, -12, -11, 8, -11, -11, 8)$ се получава: 32 се среща 1 път, -12 се среща 1 път, -11 се среща 3 пъти и 8 се среща 2 пъти. Полученият резултат да се извежда в сортиран вид на различните числа.

Решение: Нека числата са в масива `double[] numbers`. Ще формулираме два основни алгоритъма за решаване на задачата:

Алгоритъм 1: Създаване на масив `double[][] differentNumbers`, като масивът $\text{differentNumbers}[0]$ съдържа различните елементи на масива numbers , а $\text{differentNumbers}[1]$ – съответния им брой срещания. Сортиране на масива $\text{differentNumbers}[0]$ във възходящ ред, като съответствието се запазва.

Алгоритъм 2: Създаване на масив `double[][] differentNumbers`, като масивът $\text{differentNumbers}[0]$ съдържа различните елементи на масива numbers , сортирани във възходящ ред, а $\text{differentNumbers}[1]$ – съответния им брой срещания.



Задачи за упражнение

Задача: Да се реализира клас, съдържащ методи за създаване и запълване на едномерен числов масив:

1. С числата от 1 до n : $(1, 2, \dots, n)$
2. С четните числа между 1 и n : $(k, \dots, 2)$, като $k = n$, ако n е четно и $k = n - 1$ в противен случай
3. С първите n члена на аритметична прогресия с първи член a_1 и разлика d : (a_1, \dots, a_n)
4. С първите n числа на Фиbonacci: $f_0 = 0, f_1 = 1, f_k = f_{k-1} + f_{k-2}, k \geq 2$: (f_0, \dots, f_{n-1})
5. С цифрите на естествено число $n = d_k \dots d_0$: $(0, \dots, 0, d_k, \dots, d_0)$

Задача: Да се реализира клас, съдържащ методи за определяне на:

1. Сумата на всички елементи на едномерен числов масив
2. Средното аритметично на всички елементи на едномерен числов масив
3. Сумата на елементите с индекси от first до last на едномерен числов масив
4. Средното аритметично на елементите с индекси от first до last на едномерен числов масив

Задача: Да се състави програма, която създава таблица и извежда нейните елементи:

- a) За дадени x_1, \dots, x_n

1	1	...	1
x_1	x_2	...	x_n
...
x_1^{n-1}	x_2^{n-1}	...	x_n^{n-1}

- b) За дадени x и $n > 0$

x	x^2	...	x^n
x^{n+1}	x^{n+2}	...	x^{2n}
...
$x^{n(n-1)+1}$	$x^{n(n-1)+2}$...	x^{n^2-n}

Задача: Да се състави програма, която за дадено n запълва и извежда таблица с n реда и n стълба с числата от 1 до n^2 по спирала. Например, за $n=4$:

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

Задача: Квадратна таблица с n реда и n стълба, съдържаща числата от 1 до n^2 , е „магически квадрат”, ако сумите на числата по редове, стълбове и двата диагонала са равни. Например, за $n=3$:

4	9	2
3	5	7
8	1	6

2	9	4
7	5	3
6	1	8

Задача: Да се състави програма, която:

- a) Проверява дали дадена квадратна таблица е магически квадрат
- b) За дадено n построява магически квадрат

Задача: Квадратна таблица с n реда и n стълба, съдържаща числата от 1 до n , е „латински квадрат”, ако елементите във всеки ред и стълб са различни. Например, за $n=3$:

1	2	3
3	1	2
2	3	1

Да се състави програма, която:

- a) Проверява дали дадена квадратна таблица е латински квадрат
- b) За дадено n построява латински квадрат

Задача: Матрицата A има седлова точка в a_{ij} , ако a_{ij} е минимален елемент в i -тия ред и максимален елемент в j -тия стълб. Да се състави програма, която намира всички седлови точки на дадена матрица A .

Задача: Детерминантата D от n^2 елемента $a_{ij}, i,j \in [1;n]$, се нарича сумата от $n!$ члена:

$$\sum (-1)^{[i_1, i_2, \dots, i_n]} a_{1i_1} \cdot a_{2i_2} \cdots a_{ni_n}$$

$\forall i_1, i_2, \dots, i_n$ - пермутация на числата от 1 до n

където $[i_1, i_2, \dots, i_n]$ е броят на инверсии на пермутацията i_1, i_2, \dots, i_n на числата от 1 до n относно главната пермутация 1 2 ... n и се означава по следния начин:

$$D = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

Да се състави програма, която пресмята:

- a) Детерминантата на Смит с елементи $a_{ij} = \text{НОД}(i, j)$

$$D = \begin{vmatrix} \text{НОД}(1, 1) & \text{НОД}(1, 2) & \dots & \text{НОД}(1, n) \\ \text{НОД}(2, 1) & \text{НОД}(2, 2) & \dots & \text{НОД}(2, n) \\ \dots & \dots & \dots & \dots \\ \text{НОД}(n, 1) & \text{НОД}(n, 2) & \dots & \text{НОД}(n, n) \end{vmatrix} = \varphi(1)\varphi(2)\dots\varphi(n)$$

където $\varphi(k)$ е броят на числата по-малки от k и взаимно прости с него

- b) Детерминантата на матрицата на Вандермонд А с елементи $a_{ij} = x_j^{i-1}$

$$D = \begin{vmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \dots & \dots & \dots & \dots \\ x_1^n & x_2^n & \dots & x_n^n \end{vmatrix}$$

$$D = \prod_{1 \leq j \leq n} x_j \prod_{1 \leq i < j \leq n} (x_j - x_i)$$

- c) Детерминантата на матрицата на Коши A с елементи $a_{ij} = 1 / (x_i + y_j)$

$$D = \begin{vmatrix} 1 / (x_1 + y_1) & 1 / (x_1 + y_2) & \dots & 1 / (x_1 + y_n) \\ 1 / (x_2 + y_1) & 1 / (x_2 + y_2) & \dots & 1 / (x_2 + y_n) \\ \dots & \dots & \dots & \dots \\ 1 / (x_n + y_1) & 1 / (x_n + y_2) & \dots & 1 / (x_n + y_n) \end{vmatrix}$$

$$D = \frac{\prod_{1 \leq i,j \leq n} (x_j - x_i)(y_j - y_i)}{\prod_{1 \leq i,j \leq n} (x_i + y_j)}$$

- d) Детерминантата на комбинаторната матрица A с елементи $a_{ij} = y + \delta_{ij}x$, където $\delta_{ij}=1$, ако $i=j$ и 0 в противен случай

$$D = \begin{vmatrix} x+y & y & \dots & y \\ y & x+y & \dots & y \\ \dots & \dots & \dots & \dots \\ y & y & \dots & x+y \end{vmatrix} = x^{n-1} (x + ny)$$



Обработка на текст

Тип char

Стойностите на целочисленият тип **char** са целите неотрицателни двоични числа с дължина 16 от 000...000 до 111...111 (десетичните числа от 0 до 65535, наричани *номера*), които представят елементите на множеството Unicode [The Unicode Standard] от '\u0000' до '\uffff', включително, наричани *символи*. Първите 128 символа на Unicode са тези на множеството ASCII [American Standard Code for Information Interchange].

С изключение на коментарите, идентификаторите и константите от тип **char** и **String**, входните елементи на програмата са съставени от символи на ASCII или символи на Unicode, които съответстват на символи на ASCII.

Всяка целочислена стойност на типа **char** се *представя* като двоично число без знак и заема в паметта на компютъра поле от 16 бита.

Над стойности на типа **char** може да се прилагат *операциите* за останалите целочислени типове.

Константа от тип **char** (*символна константа*) е:

```

CharacterLiteral:
  'SingleCharacter' 
  'EscapeSequence' 

SingleCharacter:
  InputCharacter but not ' or \ 

InputCharacter:
  UnicodeInputCharacter but not CR or LF

LineTerminator:
  the ASCII LF character, also known as "newline"
  the ASCII CR character, also known as "return"
  the ASCII CR character followed by the ASCII LF character

EscapeSequence:
  \ b          /* \u0008: backspace BS */
  \ t          /* \u0009: horizontal tab HT */
  \ n          /* \u000a: linefeed LF */
  \ f          /* \u000c: form feed FF */
  \ r          /* \u000d: carriage return CR */
  \ "
  \ '
  \ \
  OctalEscape    /* \u0000 to \u00ff: from octal value */

OctalEscape:
  \ OctalDigit
  \ OctalDigit OctalDigit
  \ ZeroToThree OctalDigit OctalDigit

OctalDigit:
  0 1 2 3 4 5 6 7

ZeroToThree:
  0 1 2 3

```

Променливи от тип **char** се декларира по познатия начин.

Програма ASCII: Извежда графичните символи на ASCII и техните номера от 32 до 126, включително, дадени в Таблица 1.

```
package Characters;

public class ASCII {
    public static void main(String[] args) {
        System.out.println("ASCII CHARACTER      NUMBER");
        for (int i = 32; i <= 126; i++) {
            System.out.print("      " + (char) i);
            System.out.println("      " + i);
        }
    }
}
```

Знак	Номер										
!	32	0	48	@	64	P	80	`	96	p	112
"	33	1	49	A	65	Q	81	a	97	q	113
#	34	2	50	B	66	R	82	b	98	r	114
\$	35	3	51	C	67	S	83	c	99	s	115
%	36	4	52	D	68	T	84	d	100	t	116
&	37	5	53	E	69	U	85	e	101	u	117
'	38	6	54	F	70	V	86	f	102	v	118
(39	7	55	G	71	W	87	g	103	w	119
)	40	8	56	H	72	X	88	h	104	x	120
*	41	9	57	I	73	Y	89	i	105	y	121
+	42	:	58	J	74	Z	90	j	106	z	122
,	43	;	59	K	75	[91	k	107	{	123
-	44	<	60	L	76	\	92	l	108		124
.	45	=	61	M	77]	93	m	109	}	125
/	46	>	62	N	78	^	94	n	110	~	126
	47	?	63	O	79	_	95	o	111		

Таблица 1. ASCII Символи и техните номера

Кластьт **Character** от пакета `java.lang` предлага още възможности за обработка на стойности на типа **char** чрез дефинираните в него конструктори, методи и константи - виж <http://java.sun.com/javase/6/docs/api/>.

За *вход* на символи на ASCII или Unicode, които съответстват на символи на ASCII може да се използва методът `Console.in.read()` от пакета `ccj`, който чете един символ, въведен от клавиатурата и връща неговия номер като стойност на типа `int`, а за *изход* - методът `System.out.print(Параметър)` от класа **System** на пакета `java.lang`, който преобразува стойността на параметъра в стойност на типа **String** и я извежда в прозорец на екрана.

Тип String

Всеки екземпляр (*обект*) на класа **String** от пакета `java.lang` - виж <http://java.sun.com/javase/6/docs/api/> е последователност от символи на Unicode. Над обектите на класа **String** могат да се прилагат *методите* на класа. Декларирането на променливи от този тип става по познатия вече начин и за тях са валидни *операциите за съставни типове*.

Константа от тип **String** (*низ*) е:

*StringLiteral:
" StringCharacters_{opt} "*

*StringCharacters:
StringCharacter
StringCharacters StringCharacter*

StringCharacter:
*InputCharacter but not " or *
EscapeSequence

За *вход* на низове може да се използва методът `Console.in.readLine()` от пакета `ccj`, който чете последователност от символи до край на ред, въведена от клавиатурата и я връща като стойност на типа **String**, а за *изход* - методът `System.out.print(Параметър)` от класа `System` на пакета `java.lang`, който извежда стойността на параметъра в прозорец на екрана.

Задача: Да се състави програма, която запълва по редове квадратна таблица $A(n \times n)$, $n \leq 6$ с главните латински букви, започвайки от буквата A и след това транспонира елементите на таблицата A без използване на допълнителен масив.

Реализация:

```

package Characters;

import ccj.*;

public class Example5_1 {
    public static void main(String[] args) {
        char symbol = 'A';
        int i, j;
        System.out.print("Enter a positive integer n: 1<n<6 => ");
        int n = Console.in.readInt();
        char m[][] = new char[n][n];
        // Creation of matrix m:
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                m[i][j] = symbol++;
        // Output of matrix m:
        System.out.println("MATRIX IS:");
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++)
                System.out.print(m[i][j] + "    ");
            System.out.println("");
        }
        // Transponirane:
        for (i = 0; i < n; i++) {
            for (j = i + 1; j < n; j++) {
                symbol = m[i][j];
                m[i][j] = m[j][i];
                m[j][i] = symbol;
            }
        }
        // Output of new matrix m:
        System.out.println("NEW MATRIX IS:");
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++)
                System.out.print(m[i][j] + "    ");
            System.out.println("");
        }
    }
}

```

Изпълнение на програмата:

```

Enter a positive integer n: 1<n<6 => 5
MATRIX IS:
A   B   C   D   E
F   G   H   I   J
K   L   M   N   O
P   Q   R   S   T

```

```

U   V   W   X   Y
NEW MATRIX IS:
A   F   K   P   U
B   G   L   Q   V
C   H   M   R   W
D   I   N   S   X
E   J   O   T   Y

```

Задача: Да се състави програма, която чете текст и извършва следните обработки:

- Определя броя на редовете в текста
- Определя броя на графичните символи в текста
- Определя колко пъти се среща всяка от малките латински букви в текста
- Извежда текста

Реализация:

```

package Characters;

import ccj.*;

public class Example5_2 {
    public static void main(String[] args) {
        int c;
        int lines = 0, symbols = 0;
        int count[] = new int[26];
        for (int i = 0; i < 26; i++)
            count[i] = 0;
        // Text input:
        while ((c = Console.in.read()) != -1) {
            if (c == '\r') {
                lines++; // line
                System.out.println("");
            } else if ((' ' <= c) && (c <= '~')) {
                System.out.print((char) c); // output
                symbols++;
                if (('a' <= c) && (c <= 'z'))
                    count[c - 'a']++;
            }
        }
        // Output:
        System.out.println();
        System.out.println("Number of lines = " + lines);
        System.out.println("Number of symbols = " + symbols);
        System.out.println("Letters:");
        for (int i = 0; i < 26; i++)
            if (count[i] != 0)
                System.out.println("    " + (char) (i + 'a') + "=> "
                    + count[i]);
    }
}

```

Изпълнение на програмата:

```

This*is
This*is
a*little
a*little
example.
example.
<Ctrl-Z>
Number of lines = 3
Number of symbols = 23
Letters:
    a=> 2

```

```
e=> 3
h=> 1
i=> 3
l=> 3
m=> 1
p=> 1
s=> 2
t=> 2
x=> 1
```

Задача: Да се състави програма, която проверява дали последователност от символи, завършваща със символа #, е съставена от малки и главни латински букви, цифри, символа \$ и символа за подчертаване _ и започва с буква, \$ или _.

Програмата проверява дали дадена последователност отговаря на условията, като използва целочисленна променлива n с начална стойност 0, като стойността ѝ се променя в зависимост от поредния прочетен символ, както следва:

n	Буква, \$, _	Цифра	Символ, различен от буква, цифра, \$ и _	#
0	1	3	3	3
1	1	1	3	2

Ако след прочитане на последователността n=2, то тя отговаря на условията, а ако n=3 – не отговаря.

Реализация:

```
package Characters;

import ccj.*;

public class Example5_3 {
    public static void main(String[] args) {
        int n = 0;
        char c = ' ';
        while (c != '#') {
            c = (char) Console.in.read();
            switch (n) {
                case 0:
                    if (('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z')
                        || (c == '$') || (c == '_'))
                        n = 1;
                    else
                        n = 3;
                    break;
                case 1:
                    if (('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z')
                        || (c == '$') || (c == '_') || ('0' <= c && c <= '9'))
                        n = 1;
                    else if (c == '#')
                        n = 2;
                    else
                        n = 3;
                    break;
                case 2:
                case 3:
                    break;
            }
            c = (n == 2) ? 'Y' : 'N';
            System.out.println(c);
        }
    }
}
```

Изпълнение на програмата:

Първо изпълнение:

\$1a_s#
Y

Трето изпълнение:

1sd#
N

Пето изпълнение:

sd\$#
Y

Второ изпълнение:

N

Четвърто изпълнение:

as1_#
Y

Задача: Да се състави програма, която чете последователност от символи, завършваща със символа = и представляваща аритметичен израз баз скоби с операции +, -, *, / и операнди – цели десетични числа без знак. Пресмята стойността на израза, като изпълнява операциите отляво надясно, преобразува я в последователност от символи-цифри и я извежда.

Реализация:

```
package Characters;

import ccj.*;

public class Example5_4 {
    public static void main(String[] args) {
        int operand, base = 10, d, result = 0;
        char op = '+', c;
        while (op != '=') {
            // Second operand for operator: char==>int
            operand = 0;
            c = (char) Console.in.read();
            while (('0' <= c) && (c <= '9')) {
                operand = operand * base + (c - '0');
                c = (char) Console.in.read();
            }
            switch (op) {
                case '+':
                    result += operand;
                    break;
                case '-':
                    result -= operand;
                    break;
                case '*':
                    result *= operand;
                    break;
                case '/':
                    result /= operand;
                    break;
            }
            op = c;
        }
        // Type conversion: int==>char
        char dec[] = new char[15];
        d = -1;
        do {
            d++;
            dec[d] = (char) (result % base + '0');
            result /= base;
        } while (result != 0);
        // Output:
        for (int i = d; i >= 0; i--)
            System.out.print(dec[i]);
    }
}
```

}

Изпълнение на програмата:

Първо изпълнение:

$$12 \cdot 5 + 10 / 10 =$$

7

Второ изпълнение:

$$126 / 10 + 22 =$$

34

Задача: Да се състави програма, която:

1. Кодира даден текст, като заменя всяка малка латинска буква със съответната ѝ по място малка латинска буква от дадено множество
2. Декодира кодирания текст
3. Сравнява декодирания текст с първоначалния, за да установи дали те съвпадат или не

Реализация:

```
package Characters;

import ccj.*;

public class Example5_5 {
    public static void main(String[] args) {
        // Code:
        System.out.println("    Code:");
        for (int i = 'a'; i <= 'z'; i++)
            System.out.print((char) i);
        System.out.println();
        String code = Console.in.readLine();

        // Text:
        System.out.println("\n    Text line:");
        String text = Console.in.readLine();
        if (text.length() == 0)
            System.out.println("No text!");
        else {
            // Kodirane:
            String codedText = "";
            for (int i = 0; i < text.length(); i++) {
                char c = text.charAt(i);
                if (Character.isLowerCase(c))
                    c = code.charAt(c - 'a');
                codedText = codedText + c;
            }
            System.out.println("\n    Coded text:");
            System.out.println(codedText);

            // Dekodirane:
            String decodedText = "";
            for (int i = 0; i < codedText.length(); i++) {
                char c = codedText.charAt(i);
                for (int j = 0; j < code.length(); j++)
                    if (c == code.charAt(j)) {
                        c = (char) (j + 'a');
                        break;
                    }
                decodedText = decodedText + c;
            }
            System.out.println("\n    Decoded text:");
            System.out.println(decodedText);

            // Sravnenie:
        }
    }
}
```

```
        System.out.print("\n    Test: ");
        System.out.println(text.compareTo(decodedText) == 0 ? "Yes" : "No");
    }
}
```

Изпълнение на програмата:

Code:
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyz

Text line:
This is an example.

Coded text:
Trsc sc kx ohkwzvo.

Decoded text:
This is an example.

Test: Yes



Задачи за упражнение

Задача: Да се състави програма, която „превежда” от английски на френски

- Имената на цветовете
- Имената на месеците
- Имената на дните

Задача: Да се състави програма, която чете дата във вида *Номер_на_ден.Номер_на_месец.Година*. Проверява дали това е работен или почивен ден. Определя датата на следващия ден и я извежда във вида *Име_на_месец, Номер_на_ден, Година*.

Задача: Да се състави програма, която премахва всички коментари от програма на Java, започващи с //.

Задача: Да се състави програма, която заменя в даден текст всяка малка латинска буква със съответната ѝ главна латинска буква.

Задача: Да се състави програма, която чете две думи word и newWord и заменя в даден текст всяка дума word с думата newWord.

Задача: Да се състави програма, която определя и извежда в азбучен ред различните думи в даден текст и колко пъти всяка една от тях се среща в текста.

Задача: Даден е списък S от думи, наредени в азбучен ред. Да се състави програма, която за всяка дума W от даден текст проверява дали W се среща в S по метода на двоичното търсене. В случай, че W не бъде намерена в S, програмата я извежда.

Задача: Да се състави клас, който съдържа:

- Метод public static boolean isIdentifier(String arg), който проверява дали arg е идентификатор
- Метод public static boolean isInteger(String arg), който проверява дали arg е цяло десетично число без знак

- Метод `public static boolean isFloat(String arg)`, който проверява дали `arg` е реално число с експонента от вида *Мантиса Е Порядък*
- Метод `public static boolean isFor(String arg)`, който проверява дали `arg` има вида `for(И1=Цяло_число;И2<Цяло_число;И3++)`, като `И1`, `И2` и `И3` са един и същ идентификатор
- Метод `public static boolean isWhile(String arg)`, който проверява дали `arg` има вида `while(Идентификатор<Реално_число)`

Задача: Да се състави програма, която пресмята средното аритметично на всички цели десетични числа без знак, съдържащи се в даден текст.

Задача: Да се състави клас, който съдържа методи на клас за кодиране и декодиране на даден текст. Всеки метод да има параметър от тип **Stirng** - текста, който трябва да се кодира или декодира и да връща преобразувания текст, като стойност на типа **Stirng**. За кодиране на текста да се избере един от посочените по-долу начини:

1. Всеки символ от текста се заменя със съответния си номер с дължина три.
2. Всяка малка латинска буква се заменя с третата след нея в азбуката, т.е. а се заменя с d, b се заменя с e, ..., w се заменя със z, x се заменя с a, у се заменя с b и z се заменя със символа с
3. Всяка двойка съседни малки латински букви I и J се заменя с I,J-тия елемент на дадена таблица T(26x26) от символи

Задача: Да се състави клас, който съдържа методи на клас за четене, извеждане, събиране и умножение на големи цели положителни десетични числа.



Програмиране с използване на рекурсия

Рекурсията като метод за решаване на задачи се състои в:

1. *Параметризация на задачата* – определяне на параметрите, от които зависи решението на задачата
2. *Формулиране на тривиалните (базовите) случаи*, които се решават без използване на рекурсия
3. *Декомпозиция на задачата* на една или повече задачи от същия вид с променени параметри, които ги приближават към тривиалните случаи или съвпадат с тях

Задача: Да се реализира и тества клас за рекурсивно пресмятане на произведението на две цели числа m и n>0.

Решение: Рекурсивната дефиниция на функцията $f(m,n)=mn$ е:

$$f(m,n) = \begin{cases} m, & \text{ако } n=1 \\ m+f(m,n-1), & \text{ако } n>1 \end{cases}$$

Реализация:

```
package Recursion;

import java.util.Scanner;

public class Calculator {
    public int counter;

    public int mult(int m, int n) {
        counter++;
        for (int i = 0; i < counter; i++)
            System.out.print(" ");
        System.out.print("Counter = " + counter);
        System.out.println(" M = " + m + " N = " + n);
        int result;
        if (n == 1)
            result = m;
        else
            result = m + mult(m, n - 1);
        for (int i = 0; i < counter; i++)
            System.out.print(" ");
        System.out.print("Counter = " + counter);
        System.out.println(" Result = " + result);
        counter--;
        return result;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter an integer m: ");
        int m = sc.nextInt();
        System.out.print("Enter an integer n>0: ");
        int n = sc.nextInt();
        Calculator calc = new Calculator();
        calc.counter = 0;
        System.out.println(m + " * " + n + " = " + calc.mult(m, n));
    }
}
```

Резултати от тестването:

```

Enter an integer m: 6
Enter an integer n>0: 5
Counter = 1 M = 6 N = 5
Counter = 2 M = 6 N = 4
Counter = 3 M = 6 N = 3
Counter = 4 M = 6 N = 2
Counter = 5 M = 6 N = 1
Counter = 5 Result = 6
Counter = 4 Result = 12
Counter = 3 Result = 18
Counter = 2 Result = 24
Counter = 1 Result = 30
6 * 5 = 30

```

Задача: Да се реализира и тества клас за рекурсивна проверка дали число се среща в сортиран във възходящ ред числов масив по *метода на двоичното търсене*.

Решение: Рекурсивната дефиниция на функцията `isMember(arg,left,right,t)`, която има стойност `true`, ако `t` съвпада с някой от елементите $arg[left] \leq arg[left+1] \leq \dots \leq arg[right]$ и `false` в противен случай, е:

$$isMember(arg, left, right, t) = \begin{cases} \text{false, ако } right < left \\ \text{true, ако } right \geq left \text{ и } arg[(left+right)/2] = t \\ isMember(arg, (left+right)/2+1, right, t), \text{ ако } right \geq left \text{ и } arg[(left+right)/2] < t \\ isMember(arg, left, (left+right)/2-1, t), \text{ ако } right \geq left \text{ и } arg[(left+right)/2] > t \end{cases}$$

Реализация:

```

package Recursion;

import java.util.Scanner;

public class BinarySearcher {
    public boolean isMember(int arg[], int left, int right, int t) {
        if (right < left)
            return false;
        else {
            int mid = (left + right) / 2;
            if (arg[mid] < t)
                return isMember(arg, mid + 1, right, t);
            else if (arg[mid] > t)
                return isMember(arg, left, mid - 1, t);
            else
                return true;
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter sorted array");
        System.out.print("Enter n: ");
        int n = sc.nextInt();
        int a[] = new int[n];
        for (int i = 0; i < n; i++)
            a[i] = sc.nextInt();

        System.out.print("Enter k: ");
        int k = sc.nextInt();
        boolean f = new BinarySearcher().isMember(a, 0, n - 1, k);
    }
}

```

```

        System.out.println(f ? "Yes" : "No");
    }
}

```

Задача: Да се реализира и тества клас за рекурсивно обръщане на низ.

Решение: Нека $\text{arg} = c_1 \dots c_n$ е низ с дължина $d(\text{arg}) = n$. Рекурсивната дефиниция на функцията $\text{reverse}(\text{arg})$, която има стойност обратния низ $c_n \dots c_1$, е:

$$\text{reverse}(\text{arg}) = \begin{cases} \text{arg}, & \text{ако } d(\text{arg}) \leq 1 \\ c_n \text{reverse}(c_1 \dots c_{n-1}), & \text{ако } d(\text{arg}) > 1 \end{cases}$$

Реализация:

```

package Recursion;

import java.util.Scanner;

public class StringReverser {
    public String reverse(String arg) {
        if (arg.length() <= 1)
            return arg;
        else {
            char lastSymbol = arg.charAt(arg.length() - 1);
            String substr = arg.substring(0, arg.length() - 1);
            String result = lastSymbol + reverse(substr);
            return result;
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the string: ");
        String s = sc.nextLine();
        StringReverser reverser = new StringReverser();
        System.out.println("reverse(" + s + ")=" + reverser.reverse(s));
    }
}

```

Задача: Да се реализира и тества клас за итеративно и рекурсивно пресмятане на n -я член на редицата от числа на Фибоначи a_n и метод, който за всяко $n \in [\min; \max]$ извежда: n , a_n , времето за пресмятане на a_n итеративно и времето за пресмятане на a_n рекурсивно.

Реализация:

```

package Recursion;

import java.util.Scanner;

public class FibonachiTester {
    public int recFib(int n) {
        if (n == 0)
            return 0;
        else if (n == 1)
            return 1;
        else
            return recFib(n - 1) + recFib(n - 2);
    }

    public int iterFib(int n) {
        if (n == 0)
            return 0;
        else if (n == 1)

```

```

        return 1;
    else {
        int a0 = 0, a1 = 1, a = 1;
        for (int i = 2; i <= n; i++) {
            a = a0 + a1;
            a0 = a1;
            a1 = a;
        }
        return a;
    }
}

public void test() {
    Scanner sc = new Scanner(System.in);
    System.out.println("Testing...");
    System.out.print("    min = ");
    int min = sc.nextInt();
    System.out.print("    max = ");
    int max = sc.nextInt();
    long a, t1, t2, startTime, endTime;
    FibonachiTester tester = new FibonachiTester();
    System.out.println("N\tFib(N)\tIterTime\tRecTime (milliseconds)");
    for (int n = min; n <= max; n++) {
        // IterTime
        startTime = System.currentTimeMillis();
        a = tester.iterFib(n);
        endTime = System.currentTimeMillis();
        t1 = endTime - startTime;

        // RecTime
        startTime = System.currentTimeMillis();
        a = tester.recFib(n);
        endTime = System.currentTimeMillis();
        t2 = endTime - startTime;

        System.out.println(n + "\t" + a + "\t" + t1 + "\t" + t2);
    }
    System.out.println("Done!");
}

public static void main(String[] args) {
    new FibonachiTester().test();
}
}

```

Резултати от тестването:

```

Testing...
    min = 30
    max = 40
N    Fib(N)      IterTime   RecTime (milliseconds)
30  832040       0          40
31  1346269      0          50
32  2178309      0          130
33  3524578      0          160
34  5702887      0          351
35  9227465      0          420
36  14930352     0          751
37  24157817     0          1182
38  39088169     0          1923
39  63245986     0          3034
40  102334155    0          5087
Done!

```

Задача: Да се реализира и тества клас за рекурсивно пресмятане на стойността на функцията на Акерман $A(m,n)$ за дадени стойности на параметрите m и n – цели неотрицателни числа., съгласно нейната дефиниция:

$$A(m,n) = \begin{cases} n+1, & \text{ако } m=0 \\ A(m-1,1), & \text{ако } m \neq 0 \text{ и } n=0 \\ A(m-1,A(m,n-1)), & \text{ако } m \neq 0 \text{ и } n \neq 0 \end{cases}$$

За някои стойности на m има формули, които може да се използват при тестването:

$$\begin{array}{ll} A(0,n)=n+1 & A(3,n)=2^{n+3}-3 \\ A(1,n)=n+2 & \\ A(2,n)=2n+3 & A(4,n)=2^2-3, \text{ } n+2 \text{ вложени степени} \end{array}$$

Реализация:

```
package Recursion;

public class AkermanCalculator {
    public int akerman(int m, int n) {
        if (m == 0)
            return n + 1;
        else if (n == 0)
            return akerman(m - 1, 1);
        else
            return akerman(m - 1, akerman(m, n - 1));
    }

    public static void main(String[] args) {
        AkermanCalculator calc = new AkermanCalculator();
        System.out.println("A(0,3)=" + calc.akerman(0, 3));
        System.out.println("A(1,10)=" + calc.akerman(1, 10));
        System.out.println("A(2,11)=" + calc.akerman(2, 11));
        System.out.println("A(3,2)=" + calc.akerman(3, 2));
    }
}
```

Резултати от тестването:

```
A(0,3)=4
A(1,10)=12
A(2,11)=25
A(3,2)=29
```

Задача: Да се реализира и тества клас за рекурсивно генериране на всички думи с дадена дължина над дадена азбука.

Решение: Нека $X=\{x_1, \dots, x_n\}$ е множество, чито елементи ще наричаме букви, а самото X – азбука и $\epsilon \notin X$. Дума над азбуката X е:

1. ϵ е дума над X с дължина $d(\epsilon)=0$
2. За всяка дума ω над X ωx_i е дума над X , за всяко $i \in [1;n]$, с дължина $d(\omega x_i)=d(\omega)+1$
3. Няма други думи над X

Реализация:

```
package Recursion;

import java.util.Scanner;
```

```

public class WordsGenerator {
    public void words(String alphabet, String word, int length) {
        if (word.length() == length)
            System.out.println(word);
        else
            for (int i = 0; i < alphabet.length(); i++)
                words(alphabet, word + alphabet.charAt(i), length);
    }

    public void words(String alphabet, int length) {
        words(alphabet, "", length);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the alphabet: ");
        String a = sc.nextLine();
        System.out.print("Enter the length: ");
        int n = sc.nextInt();
        System.out.println("The words are:");
        new WordsGenerator().words(a, "", n);
    }
}

```

Резултати от тестването:

Първо изпълнение:

```

Enter alphabet: ab
Enter the length: 3
The words are:
aaa
aab
aba
abb
baa
bab
bba
bbb

```

Второ изпълнение:

```

Enter alphabet: abc
Enter the length: 2
The words are:
aa
ab
ac
ba
bb
bc
ca
cb
cc

```

Задача: Да се реализира и тества клас за рекурсивно решаване на задачата за Ханойските кули: да се преместят n диска, разположени на основа А в намаляващ ред на техните диаметри (във вид на пирамида), на основа В, като се използва междинна основа С. Единственото разрешено действие е вземане на диск на върха на пирамида и поставянето му върху диск с по-голям диаметър на друга пирамида или върху друга празна основа.

Решение: Задачата има решение за всяко цяло число $n \geq 1$ и то се състои в изпълнението на крайна последователност от инструкции, всяка от които има вида: „Вземи диск от върха на основа α и го постави на върха на пирамидата на основа β (β може да е празна основа)”, $\alpha \neq \beta$, $\alpha, \beta \in \{A, B, C\}$, т.е. $hanoi(n, A, B, C) = I_1, \dots, I_p$. Очевидно съществува цяло число $q \geq 1$, такова че:

1. Изпълнението на I_1, \dots, I_q води до преместването на $n-1$ диска от основа А на основа С с междинна основа В
2. I_{q+1} е инструкцията за вземане на най-големия диск от основа А и поставянето му на основа В
3. Изпълнението на I_{q+2}, \dots, I_p води до преместването на $n-1$ диска от основа С на основа В с междинна основа А

Следователно, рекурсивната дефиниция на процедурата е:

$$hanoi(n, A, B, C) = hanoi(n-1, A, C, B), I_{q+1}, hanoi(n-1, C, B, A), \text{ако } n > 0$$

Ако $n=0$, не се извършва действие и това е базовият случай.

Броят на преместванията p на n диска е $f(n)=2^n-1$ и не може да се намали.

$$f(n) = \begin{cases} 0, & \text{ако } n=0 \\ 2f(n-1)+1, & \text{ако } n>0 \end{cases}$$

Реализация: Броят на преместванията за дадено n се получава в променливата counter.

```
package Recursion;

import java.util.Scanner;

public class HanoiPlayer {
    public int counter;

    public void hanoi(int n, char a, char b, char c) {
        if (n > 0) {
            hanoi(n - 1, a, c, b);
            counter++;
            System.out.println("disk " + n + " from " + a + " to " + b);
            hanoi(n - 1, c, b, a);
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("n = ");
        HanoiPlayer player = new HanoiPlayer();
        int n = sc.nextInt();
        player.counter = 0;
        player.hanoi(n, 'A', 'B', 'C');
        System.out.println("Counter = " + player.counter);
    }
}
```

Резултати от тестването:

Първо изпълнение:

```
n = 3
disk 1 from A to B
disk 2 from A to C
disk 1 from B to C
disk 3 from A to B
disk 1 from C to A
disk 2 from C to B
disk 1 from A to B
Counter = 7
```

Второ изпълнение:

```
n = 4
disk 1 from A to C
disk 2 from A to B
disk 1 from C to B
disk 3 from A to C
disk 1 from B to A
disk 2 from B to C
disk 1 from A to C
disk 4 from A to B
disk 1 from C to B
disk 2 from C to A
disk 1 from B to A
disk 3 from C to B
disk 1 from A to C
disk 2 from A to B
disk 1 from C to B
Counter = 15
```

Трето изпълнение:

```
n = 5
disk 1 from A to B
disk 2 from A to C
disk 1 from B to C
disk 3 from A to B
disk 1 from C to A
disk 2 from B to C
disk 1 from A to C
disk 4 from A to B
disk 1 from C to B
disk 2 from C to A
disk 1 from B to A
disk 3 from C to B
disk 1 from A to B
disk 2 from A to C
disk 1 from B to C
disk 5 from A to B
disk 1 from C to A
disk 2 from C to B
disk 1 from A to B
disk 3 from C to A
disk 1 from B to C
disk 2 from B to A
disk 1 from C to A
```

```

disk 4 from C to B
disk 1 from A to B
disk 2 from A to C
disk 1 from B to C
disk 3 from A to B
disk 1 from C to A
disk 2 from C to B
disk 1 from A to B
Counter = 31

```

Задача: Нека M е мрежа с n реда и m стълба и $C=\{c_1, \dots, c_s\}$ е множество от $s \geq 2$ цвята. Всяко квадратче на мрежата $K(x,y)$, $x \in [0; n-1]$ и $y \in [0; m-1]$, е оцветено в един от дадените цветове. Нека $\text{color} \in C$. Квадратчетата в цвят color са “проходими”, а останалите – “непроходими”. От проходимо квадратче може да се премине във всяко от четирите му съседни, като се прекоси общата им страна. В непроходимо квадратче може да се влезе, но не може да се излезе. Последователност от различни квадратчета на мрежата q_1, \dots, q_p се нарича *път от квадратче q_1 до квадратче q_p с дължина $p-1$* , ако:

1. Всяко квадратче е проходимо
2. За всяко $k \in [1; p-1]$ квадратчетата q_k и q_{k+1} са съседни

Да се реализира и тества клас за рекурсивна проверка дали има път между две квадратчета в мрежата M .

Решение: Нека $A(x,y)$ и $B(c,d)$ са квадратчета на мрежата M и $A_1(x,y+1)$, $A_2(x,y-1)$, $A_3(x-1,y)$ и $A_4(x+1,y)$ са четирите съседни на A квадратчета. Има път $P(A,B)$ от A до B , ако има път $P(A_1,B)$, или има път $P(A_2,B)$, или има път $P(A_3,B)$, или има път $P(A_4,B)$.

Път от A до B може да се търси, прилагайки метода *търсене с връщане*, съгласно който пътят се конструира по следния начин: нека $P(A,a_r)=a_1, a_2, \dots, a_r$, където $a_1=A$ и $r \geq 1$, е намерената в даден момент част от път, като всяко от квадратчетата в него е обявено за “избрано”. Ако може да се избере неизбрано съседно квадратче a_{r+1} на a_r , то пътят се продължава с a_{r+1} , като квадратчето се обявява за “избрано”, за да не се избере повторно. В противен случай се връщаме назад до първото квадратче, от което може да се продължи пътят. Процесът спира при достигане на квадратчето B или при невъзможност пътят да бъде продължен.

Рекурсивната дефиниция на функцията $\text{isConnected}(x,y,c,d,M,\text{prohodimo})$, която има стойност true , ако има път от квадратче $A(x,y)$ до квадратче $B(c,d)$ в мрежа M , като prohodimo е стойност за проходимо квадратче и false в противен случай, е:

$$\text{isConnected}(x,y,c,d,M,\text{prohodimo}) = \begin{cases} \text{false , ако } A \notin M \text{ или } A \in M \text{ и } A \text{ е} \\ \text{непроходимо или е избрано} \\ \\ \text{true , ако } A \in M \text{ и е проходимо, но не е} \\ \text{избрано и } A=B \\ \\ \text{isConnected}(x+1,y,c,d,M,,\text{prohodimo}) \text{ или} \\ \text{isConnected}(x-1,y,c,d,M,\text{prohodimo}) \text{ или} \\ \text{isConnected}(x,y-1,c,d,M,\text{prohodimo}) \text{ или} \\ \text{isConnected}(x,y+1,c,d,M,\text{prohodimo}), \text{ като} \\ \text{A се обявява за избрано, ако} \\ A \in M \text{ и е проходимо, но не е} \\ \text{избрано и } A \neq B \end{cases}$$

Реализация: Мрежата е представена чрез масив от цели положителни числа, като цвят c_i се представя с числото i , за всяко $i \in [1; s]$. Квадратче се обявява за “избрано”, като се прави непроходимо (число, непринадлежащо на интервала $[1; s]$).

```

package Recursion;

public class Path {
    public boolean isConnected(int x, int y, int c, int d, int[][] M,
                               int prohodimo) {
        if (x >= 0 && x < M.length && y >= 0 && y < M[0].length
            && M[x][y] == prohodimo) {
            if (x == c && y == d)
                return true;
            else {
                M[x][y] = 0;
                boolean f = isConnected(x + 1, y, c, d, M, prohodimo)
                           || isConnected(x - 1, y, c, d, M, prohodimo)
                           || isConnected(x, y - 1, c, d, M, prohodimo)
                           || isConnected(x, y + 1, c, d, M, prohodimo);
                M[x][y] = prohodimo;
                return f;
            }
        } else
            return false;
    }

    public boolean isConnected(int x, int y, int c, int d, int[][] M) {
        return isConnected(x, y, c, d, M, M[x][y]);
    }

    public static void main(String[] args) {
        System.out.println("Not implemented!");
    }
}

```



Задачи за упражнение

Задача: Да се реализира и тества клас за итеративно и рекурсивно пресмятане на x^n , x - реално число, n - цяло число.

Задача: Да се реализира и тества клас за итеративно и рекурсивно пресмятане на най-големия общ делител на две цели положителни числа.

Задача: Да се реализира и тества клас за:

- Итеративна и рекурсивна проверка дали число се среща в масив по *метода на последователното търсене*
- Итеративно и рекурсивно пресмятане на сумата на елементите на едномерен масив от числа
- Итеративна и рекурсивна проверка дали два масива съвпадат

Задача: Да се реализира и тества клас за итеративна и рекурсивна проверка дали низ е палиндром.

Задача: Да се реализира и тества клас за рекурсивно четене на низ символ по символ и извеждане на символите в ред обратен на тяхното въвеждане.

Задача: Да се реализира и тества клас за рекурсивно извеждане на двоичното представяне на цяло положително число.

Задача: Редицата от полиноми $T_0(x), T_1(x), \dots, T_{n+1}(x), \dots$ се получава съгласно формулите:

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), n \geq 1$$

Да се реализира и тества клас за:

- a) Рекурсивно определяне на полинома $T_n(x)$ за дадено n
- b) Рекурсивно пресмятане на стойността на $T_n(x)$ за дадени n и x

Задача: Да се реализира и тества клас за обработка на оцветена мрежа, съдържащ:

- a) Рекурсивен метод за проверка има ли едноцветен път между две квадратчета в мрежата
- b) Рекурсивен метод за определяне на едноцветен път между две квадратчета в мрежата
- c) Рекурсивен метод за определяне на всички едноцветни пътища между две квадратчета в мрежата
- d) Рекурсивен метод за определяне на едноцветен път с минимална дължина между две квадратчета в мрежата

Задача: Да се реализира и тества клас за обработка на оцветена мрежа, съдържащ:

- a) Рекурсивен метод за определяне на размера (в брой квадратчета) на максималната едноцветна област, съдържаща дадено квадратче
- b) Рекурсивен метод за определяне на максималния размер на едноцветна област в мрежата

Задача: Да се реализира и тества клас за рекурсивно генериране на:

- a) Всички вариации от k -ти клас от n елемента
- b) Всички вариации с повторение от k -ти клас от n елемента
- c) Всички пермутации от n елемента
- d) Всички подмножества на дадено множество

Задача: Да се реализира и тества клас за решаване на задачата за разходката на коня: тръгвайки от избрано поле на шахматната дъска, конят посещава всяко от останалите полета и то точно по един път. Класът да съдържа:

- a) Рекурсивен метод, който получава произволно такова обхождане
- b) Рекурсивен метод, който получава всички такива обхождания

Задача: Да се реализира и тества клас за решаване на задачата за 8-те царици: 8 различно царици трябва да се разположат върху шахматна дъска 8x8, така че никои две да не се бият, т.е. във всяка хоризонтала, във всяка вертикалa и във всеки диагонал трябва да има само по една царица. Класът да съдържа:

- a) Рекурсивен метод, който получава произволно такова разполагане
- b) Рекурсивен метод, който получава всички такива разполагания



Обектно-ориентирано програмиране

Обектно-ориентираното програмиране се основава на следните основни концепции: *обекти, класове, наследяване и полиморфизъм*.

Обекти и класове

В Java обектите се явяват **екземпляри на класове**. Всеки клас има *спецификация* („видима“ част), представяща *интерфейса* – множеството от методи, които може да се извикват за обектите на класа и *реализация* („скрита“ част).

Пример: Клас **Point** за представяне на точка:

Спецификация:

```
package Shapes;

public class Point {
    //Декларации на променливи за данните на точка - координати x и y

    //Конструктор
    public Point(double x, double y) { ... } //Създава точка с координати x и y

    //Методи
    public double getX() { ... } //Връща координатата x на текущата точка
    public double getY() { ... } //Връща координатата y на текущата точка
    public double dest(Point p) { ... } //Определя разстоянието между текущата точка и точката p
}
```

Реализация:

```
package Shapes;

public class Point {
    // Data
    private double x, y;

    // Constructor
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // Public methods
    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public double dest(Point p) {
        return Math.sqrt(Math.pow(x - p.x, 2) + Math.pow(y - p.y, 2));
    }
}
```

Пример: Клас **Triangle** за представяне на геометрична фигура триъгълник:

Спецификация:

```

package Shapes;

public class Triangle {
    //Декларации на променливи за данните на триъгълник - върхове d1(x1,y1), d2(x2,y2), d3(x3,y3)

    //Конструктори
    public Triangle(double x1, double y1, double x2, double y2, double x3,
                   double y3) throws RuntimeException { ... }

    //Проверява дали точките d1(x1,y1), d2(x2,y2), d3(x3,y3) са върхове на триъгълник. Ако са
    //върхове, създава триъгълника. В противен случай активира изключение RuntimeException

    public Triangle(Point d1, Point d2, Point d3) throws RuntimeException { ... }

    //Проверява дали точките d1, d2, d3 са върхове на триъгълник. Ако са върхове,
    //създава триъгълника. В противен случай активира изключение RuntimeException

    //Методи
    public double area() { ... } //Определя лицето на текущия триъгълник
    public double p() { ... } //Определя периметъра на текущия триъгълник
    public boolean isMember(Point p) { ... } //Проверява дали точката p принадлежи на текущия триъгълник
}

```

Реализация:

```

package Shapes;

public class Triangle {
    // Data
    private double[] v;

    // Constructors
    public Triangle(double x1, double y1, double x2, double y2, double x3,
                   double y3) throws RuntimeException {
        if ((x3 - x1) * (y2 - y1) - (y3 - y1) * (x2 - x1) == 0)
            throw new RuntimeException("Invalid data!");
        v = new double[6];
        v[0] = x1;
        v[1] = y1;
        v[2] = x2;
        v[3] = y2;
        v[4] = x3;
        v[5] = y3;
    }

    public Triangle(Point d1, Point d2, Point d3) throws RuntimeException {
        this(d1.getX(), d1.getY(), d2.getX(), d2.getY(), d3.getX(), d3.getY());
    }

    // Public methods
    public double area() {
        double a = (v[2] - v[0]) * (v[2] - v[0]) + (v[3] - v[1])
                  * (v[3] - v[1]);
        double b = (v[2] - v[4]) * (v[2] - v[4]) + (v[3] - v[5])
                  * (v[3] - v[5]);
        double c = (v[0] - v[4]) * (v[0] - v[4]) + (v[1] - v[5])
                  * (v[1] - v[5]);

        a = Math.sqrt(a);
        b = Math.sqrt(b);
        c = Math.sqrt(c);
        double p = (a + b + c) / 2.0;
        return Math.sqrt(p * (p - a) * (p - b) * (p - c));
    }
}

```

```

public double p() {
    System.out.println("Not implemented!");
    return 0;
}

public boolean isMember(Point p) {
    System.out.println("Not implemented!");
    return false;
}
}

```

Тестване:

```

package Shapes;

import java.util.Scanner;

public class TestTriangle {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter x1: ");
        double x1 = sc.nextDouble();
        System.out.print("Enter y1: ");
        double y1 = sc.nextDouble();
        System.out.print("Enter x2: ");
        double x2 = sc.nextDouble();
        System.out.print("Enter y2: ");
        double y2 = sc.nextDouble();
        System.out.print("Enter x3: ");
        double x3 = sc.nextDouble();
        System.out.print("Enter y3: ");
        double y3 = sc.nextDouble();

        Triangle t = new Triangle(x1, y1, x2, y2, x3, y3);
        System.out.println("Area = " + t.area());
    }
}

```

Резултати от тестването:

Първо изпълнение

```

Enter x1: 0
Enter y1: 0
Enter x2: 0
Enter y2: 5
Enter x3: 5
Enter y3: 0
Area = 12.5

```

Второ изпълнение

```

Enter x1: 1
Enter y1: 0
Enter x2: 7
Enter y2: 0
Enter x3: 3
Enter y3: 0
Exception in thread "main" java.lang.RuntimeException: Invalid data!
at Shapes.Triangle.<init>(Triangle.java:13)
at Shapes.TestTriangle.main(TestTriangle.java:21)

```

Пример: Промяната на реализациите на клас не води до промяна на класовете, които използват класа:

```

package Shapes;

public class Triangle {
    // Data
    private Point d1;
    private Point d2;
    private Point d3;

    // Constructors
    public Triangle(double x1, double y1, double x2, double y2, double x3,
                    double y3) throws RuntimeException {
        if ((x3 - x1) * (y2 - y1) - (y3 - y1) * (x2 - x1) == 0)
            throw new RuntimeException("Invalid data!");
        d1 = new Point(x1, y1);
        d2 = new Point(x2, y2);
        d3 = new Point(x3, y3);
    }

    public Triangle(Point d1, Point d2, Point d3) throws RuntimeException {
        this(d1.getX(), d1.getY(), d2.getX(), d2.getY(), d3.getX(), d3.getY());
    }

    // Public methods
    public double area() {
        double a = d1.dist(d2);
        double b = d2.dist(d3);
        double c = d1.dist(d3);
        double p = (a + b + c) / 2.0;
        return Math.sqrt(p * (p - a) * (p - b) * (p - c));
    }

    public double p() {
        System.out.println("Not implemented!");
        return 0;
    }

    public boolean isMember(Point p) {
        System.out.println("Not implemented!");
        return false;
    }
}

```

Тестване:

```

package Shapes;

import java.util.Scanner;

public class TestTriangle {
    // Тялото не се променя
}

```

Резултати от тестването:

Първо изпълнение

```

Enter x1: 0
Enter y1: 0
Enter x2: 0
Enter y2: 5
Enter x3: 5
Enter y3: 0
Area = 12.5

```

Второ изпълнение

```
Enter x1: 1
Enter y1: 0
Enter x2: 7
Enter y2: 0
Enter x3: 3
Enter y3: 0
Exception in thread "main" java.lang.RuntimeException: Invalid data!
  at Shapes.Triangle.<init>(Triangle.java:11)
  at Shapes.TestTriangle.main(TestTriangle.java:21)
```

Пример: Разрешаването на достъпа може да доведе до неправомерна промяна на данните на обект:

```
package Shapes;

public class Triangle {
    // Data
    public Point d1;
    public Point d2;
    public Point d3;

    // Constructors
    public Triangle(double x1, double y1, double x2, double y2, double x3,
                    double y3) throws RuntimeException {
        if ((x3 - x1) * (y2 - y1) - (y3 - y1) * (x2 - x1) == 0)
            throw new RuntimeException("Invalid data!");
        d1 = new Point(x1, y1);
        d2 = new Point(x2, y2);
        d3 = new Point(x3, y3);
    }

    public Triangle(Point d1, Point d2, Point d3) throws RuntimeException {
        this(d1.getX(), d1.getY(), d2.getX(), d2.getY(), d3.getX(), d3.getY());
    }

    // Public methods
    public double area() {
        double a = d1.dist(d2);
        double b = d2.dist(d3);
        double c = d1.dist(d3);
        double p = (a + b + c) / 2.0;
        return Math.sqrt(p * (p - a) * (p - b) * (p - c));
    }

    public double p() {
        System.out.println("Not implemented!");
        return 0;
    }

    public boolean isMember(Point p) {
        System.out.println("Not implemented!");
        return false;
    }
}
```

Тестване:

```
package Shapes;

import java.util.Scanner;

public class TestTriangle {
    public static void main(String[] args) {
```

```

Scanner sc = new Scanner(System.in);
System.out.print("Enter x1: ");
double x1 = sc.nextDouble();
System.out.print("Enter y1: ");
double y1 = sc.nextDouble();
System.out.print("Enter x2: ");
double x2 = sc.nextDouble();
System.out.print("Enter y2: ");
double y2 = sc.nextDouble();
System.out.print("Enter x3: ");
double x3 = sc.nextDouble();
System.out.print("Enter y3: ");
double y3 = sc.nextDouble();

Triangle t = new Triangle(x1, y1, x2, y2, x3, y3);
System.out.println("Area = " + t.area());

t.d3 = new Point(0.0, 7.0);
System.out.println("Area = " + t.area());
}
}

```

Резултати от тестването:

```

Enter x1: 0
Enter y1: 0
Enter x2: 0
Enter y2: 5
Enter x3: 5
Enter y3: 0
Area = 12.5
Area = 0.0

```

Пример: Клас **Block** за представяне на геометрична фигура правоъгълник:

Спецификация:

```

package Shapes;

public class Block {
    //Декларации на променливи за данните на правоъгълник - горен ляв връх d1(x1,y1) и
    //долен десен връх d2(x2,y2)

    //Конструтори
    public Block (double x1, double y1, double x2, double y2) throws RuntimeException {...}
    //Проверява дали точките d1(x1,y1), d2(x2,y2) са върхове на правоъгълник. Ако са върхове,
    //създава правоъгълника. В противен случай активира изключение RuntimeException

    public Block (Point d1, Point d2) throws RuntimeException {...}
    //Проверява дали точките d1, d2 са върхове на правоъгълник. Ако са върхове,
    //създава правоъгълника. В противен случай активира изключение RuntimeException

    //Методи
    public double area() {...}//Определя лицето на текущия правоъгълник
    public double p() {...}//Определя периметъра на текущия правоъгълник
    public boolean isMember(Point p) {...}//Проверява дали точката p принадлежи на текущия правоъгълник
}

```

Реализация:

```

package Shapes;

public class Block extends Shape {

```

```

// Data
private Point d1; // top left corner
private Point d2; // down right corner

// Constructors
public Block(double x1, double y1, double x2, double y2)
    throws RuntimeException {
    if (x1 == x2 || y1 == y2)
        throw new RuntimeException("Invalid data!");
    d1 = new Point(x1, y1);
    d2 = new Point(x2, y2);
}

public Block(Point d1, Point d2) throws RuntimeException {
    this(d1.getX(), d1.getY(), d2.getX(), d2.getY());
}

// Public methods
public double area() {
    Point m = new Point(d1.getX(), d2.getY());
    return m.dist(d1) * m.dist(d2);
}

public double p() {
    Point m = new Point(d1.getX(), d2.getY());
    return 2 * (m.dist(d1) + m.dist(d2));
}

public boolean isMember(Point p) {
    return d1.getX() <= p.getX() && p.getX() <= d2.getX()
        && d2.getY() <= p.getY() && p.getY() <= d1.getY();
}
}

```

Декларацията на клас в Java може да съдържа ключовата дума `abstract`, с което се забранява създаването на екземпляри на класа.

Декларацията на метод-член на клас може да съдържа ключовата дума `abstract`, което означава, че методът не се реализира в класа. Ако клас съдържа поне един абстрактен метод, то самият клас също е абстрактен.

Пример: Абстрактен клас `Shape`:

Спецификация:

```
package Shapes;
```

```

public abstract class Shape {
    public abstract double area(); // Определя лицето на текущата фигура
    public abstract double p(); // Определя периметъра на текущата фигура
    public abstract boolean isMember(Point p); // Проверява дали точката p принадлежи на текущата фигура
}

```

Реализация:

```

package Shapes;

public abstract class Shape {
    public abstract double area();
    public abstract double p();
    public abstract boolean isMember(Point p);
}

```

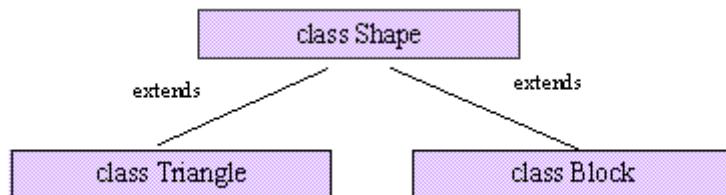
Наследяване и полиморфизъм

В Java клас може да има само един надклас. Декларацията на подклас съдържа ключовата дума `extends` и името на надкласа. Ако декларацията на клас съдържа ключовата дума `final`, то той не може да има подкласове.

Ако декларацията на метод-член на клас съдържа ключовата дума `final`, то методът не може да се предефинира в подклас.

На променлива от тип клас може да се присвоява адрес на екземпляр на негов подклас.

Пример: Йерархията:



```

public abstract class Shape {
    //Тялото не се променя
}
  
```

Класовете **Triangle** и **Block** реализират абстрактните методи на класа **Shape**:

```

public class Triangle extends Shape {
    //Тялото не се променя
}

public class Block extends Shape {
    //Тялото не се променя
}
  
```

Тестване:

```

package Shapes;

import java.util.Scanner;

public class TestShape {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("      Triangle:");
        System.out.print("Enter x1: ");
        double x1 = sc.nextDouble();
        System.out.print("Enter y1: ");
        double y1 = sc.nextDouble();
        System.out.print("Enter x2: ");
        double x2 = sc.nextDouble();
        System.out.print("Enter y2: ");
        double y2 = sc.nextDouble();
        System.out.print("Enter x3: ");
        double x3 = sc.nextDouble();
        System.out.print("Enter y3: ");
        double y3 = sc.nextDouble();
        Triangle t = new Triangle(x1, y1, x2, y2, x3, y3);

        System.out.println("      Block:");
        System.out.print("Enter x1: ");
        x1 = sc.nextDouble();
        System.out.print("Enter y1: ");
  
```

```

y1 = sc.nextDouble();
System.out.print("Enter x2: ");
x2 = sc.nextDouble();
System.out.print("Enter y2: ");
y2 = sc.nextDouble();
Block b = new Block(x1, y1, x2, y2);

Shape s;
s = t;
System.out.println("Area = " + s.area());

s = b;
System.out.println("Area = " + s.area());
}
}

```

Резултати от тестването:

```

Triangle:
Enter x1: 0
Enter y1: 0
Enter x2: 0
Enter y2: 5
Enter x3: 5
Enter y3: 0
Block:
Enter x1: 5
Enter y1: 0
Enter x2: 0
Enter y2: 5
Area = 12.5
Area = 25.0

```

В Java свързването на извикания метод с конкретна реализация става по време на изпълнение - *динамично свързване*, при което изборът е в зависимост от типа на обекта, чито адрес се съдържа в променливата. Например, `s.area()`: по време на изпълнение се прави проверка за типа на обекта с адрес в `s`. Ако типът е **Triangle** се изпълнява методът `area()` на класа **Triangle**. Ако типът е **Block**, се изпълнява методът `area()` на класа **Block**.

В езика Java може да се дефинира протокол за взаимодействие между два обекта, принадлежащи на един и същи клас или на различни класове, чрез декларация на **интерфейс**, която се състои от следните компоненти:

- Спецификатор за достъп
- Ключовата дума `interface`
- Име на интерфейса
- Ключовата дума `extends` – интерфейсът е подинтерфейс
- Имена на родителски интерфейси
- Тяло на интерфейса

Тялото на интерфейса може да съдържа декларации на константи и спецификации на методи, като всяка спецификация завършва със символа ; (методи без тела). Всеки метод се счита `public` и `abstract`, а всяка константа – `public` и `static`.

Един или повече класове могат да реализират един или повече интерфейса, с което се определя протокол за взаимодействие между обектите на тези класове. Декларацията на клас, реализиращ интерфейси, съдържа ключовата дума `implements` и списък от имената на интерфейсите.

Името на интерфейс се явва име на съставен тип и може да се използва на всяко място, на което може да се използва име на тип.

Пример: Интерфейс Shape:

Спецификация:

```
package Shapes;

public interface Shape {
    public double area(); //Определя лицето на текущата фигура
    public double p(); //Определя периметъра на текущата фигура
    public boolean isMember(Point p);
    //Проверява дали точката p принадлежи на текущата фигура
}
```

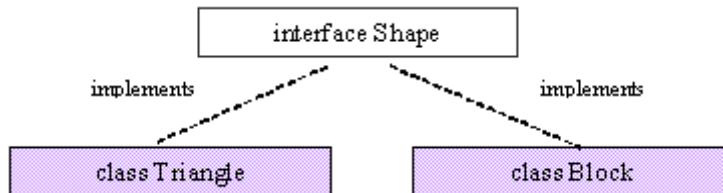
Реализация:

```
package Shapes;

public interface Shape {
    public double area();
    public double p();
    public boolean isMember(Point p);
}
```

На променлива от тип интерфейс може да се присвоява адрес само на екземпляр на клас, който реализира интерфейса.

Пример: Йерархията:



```
public interface Shape {
    //Тялото не се променя
}
```

Класовете **Triangle** и **Block** реализират интерфейса **Shape**:

```
public class Triangle implements Shape {
    //Тялото не се променя
}

public class Block implements Shape {
    //Тялото не се променя
}
```

Тестване:

```
package Shapes;

import java.util.Scanner;

public class TestShape {
    //Тялото не се променя
}
```

Резултати от изпълнението:

```

Triangle:
Enter x1: 0
Enter y1: 5
Enter x2: 0
Enter y2: 0
Enter x3: 5
Enter y3: 0
    Block:
Enter x1: 5
Enter y1: 0
Enter x2: 0
Enter y2: 5
Area = 12.5
Area = 25.0

```

Следващият пример илюстрира създаването на колекция от обекти, принадлежащи на юерархия от класове и нейната обработка.

Пример: Клас **Pattern** за представяне на линеен шаблон от фиксиран брой геометрични фигури:

Спецификация:

```

package Shapes;

public class Pattern {
    //Декларации на променливи за данните: array за обектите на шаблона, принадлежащи на юерархия
    //с корен Shape (абстрактен клас или интерфейс) и number за броя на обектите

    //Конструктор
    public Pattern(int n) { ... } //Създава празен шаблон с n места за фигури

    //Методи
    public int get() { ... } //Връща броя на фигурите в текущия шаблон
    public Shape get(int i) { ... } //Връща i-та фигура в текущия шаблон, i = 0, 1, ..., n-1
    public void insert(Shape arg) { ... } //Включва фигурата arg в текущия шаблон
    public void print() { ... } //Извежда текущия шаблон
}

```

Реализация:

```

package Shapes;

public class Pattern {
    // Data
    private Shape[] array;
    private int number;

    // Constructor
    public Pattern(int n) {
        number = 0;
        array = new Shape[n];
    }

    // Public methods
    public int size() {
        return number;
    }

    public Shape get(int i) {
        return array[i];
    }
}

```

```

}

public void insert(Shape arg) {
    array[number] = arg;
    number++;
}

public void print() {
    for (int i = 0; i < number; i++)
        System.out.println(array[i].toString());
}
}

```

Тестване:

```

package Shapes;

public class UseShapes {
    public static void main(String[] args) {
        Point point = new Point(2, 2);
        Triangle t = new Triangle(new Point(0, 0), new Point(0, 5), new Point(
            5, 0));
        Block b = new Block(new Point(0, 5), new Point(5, 0));

        Pattern p = new Pattern(10);
        p.insert(t);
        p.insert(b);
        p.insert(b);
        for (int i = 0; i < p.size(); i++) {
            Shape temp = p.get(i);
            System.out.println("Shape: " + i);
            System.out.println(" Area=" + temp.area());
            System.out.println(" P=" + temp.p());
            System.out.println(" Point is member: " + temp.isMember(point));
        }
    }
}

```

Резултати от тестването:

```

Shape: 0
Area=12.5
Not implemented!
P=0.0
Not implemented!
Point is member: false
Shape: 1
Area=25.0
P=20.0
Point is member: true
Shape: 2
Area=25.0
P=20.0
Point is member: true

```

Включването на нов вид обекти в колекцията и тяхната обработка се състои в разширяването на юерархията от класове с нов клас за представяне на обектите.

Пример: Клас **Circle** за представяне на окръжност:

Спецификация:

```
package Shapes;
```

```

public class Circle {
    //Декларации на променливи за данните на окръжност - център с и радиус г

    //Конструктори
    public Circle(double x, double y, double r) throws RuntimeException {...}
    //Проверява дали радиусът г е положително число. Ако е положително число, създава окръжността.
    //В противен случай активира изключение RuntimeException

    public Circle(Point c, double r) throws RuntimeException {...}
    //Проверява дали радиусът г е положително число. Ако е положително число, създава окръжността.
    //В противен случай активира изключение RuntimeException

    //Методи
    public double area() {...}//Определя лицето на текущата окръжност
    public double p() {...}//Определя периметъра на текущата окръжност
    public boolean isMember(Point p) {...}//Проверява дали точката p принадлежи на текущата окръжност
}

```

Реализация:

```

package Shapes;

public class Circle extends Shape {
    // Data
    private Point c;
    private double r;

    // Constructors
    public Circle(double x, double y, double r) throws RuntimeException {
        if (r <= 0)
            throw new RuntimeException("Invalid data!");
        this.c = new Point(x, y);
        this.r = r;
    }

    public Circle(Point c, double r) throws RuntimeException {
        this(c.getX(), c.getY(), r);
    }

    // Public methods
    public double area() {
        return Math.PI * r * r;
    }

    public double p() {
        return 2 * Math.PI * r;
    }

    public boolean isMember(Point p) {
        return p.dist(c) <= r;
    }
}

```

Тестване:

```

package Shapes;

public class UseShapes {
    public static void main(String[] args) {
        Point point = new Point(2, 2);
        Triangle t = new Triangle(new Point(0, 0), new Point(0, 5), new Point(5, 0));
        Block b = new Block(new Point(0, 5), new Point(5, 0));
        Circle c = new Circle(new Point(0, 0), 5);
    }
}

```

```

        Pattern p = new Pattern(10);
        p.insert(t);
        p.insert(b);
        p.insert(c);
        p.insert(b);
        p.insert(c);
        for (int i = 0; i < p.size(); i++) {
            Shape temp = p.get(i);
            System.out.println("Shape: " + i);
            System.out.println(" Area=" + temp.area());
            System.out.println(" P=" + temp.p());
            System.out.println(" Point is member: " + temp.isMember(point));
        }
    }
}

```

Резултати от тестването:

```

Shape: 0
Area=12.5
Not implemented!
P=0.0
Not implemented!
Point is member: false
Shape: 1
Area=25.0
P=20.0
Point is member: true
Shape: 2
Area=78.53981633974483
P=31.41592653589793
Point is member: true
Shape: 3
Area=25.0
P=20.0
Point is member: true
Shape: 4
Area=78.53981633974483
P=31.41592653589793
Point is member: true

```

Системата от класове в Java има дървовидна структура с корен класът **Object**. Всеки клас наследява методите:

- `clone` за създаване на копие на обект
- `equals` за проверка дали два обекта са равни
- `toString` за преобразуване на обект в стойност на типа **String**

Пример: Модификация на класа **Point**: предизвикане на наследените от класа **Object** методи `clone`, `equals` и `toString`

Резултати от тестването без методите `clone`, `equals` и `toString` да са предизвикани:

```

package Shapes;

public class TestPoint {
    public static void main(String[] args) {
        Point p = new Point(-2, 3);
        Point q = new Point(-2, 3);
        Point r = p;
        System.out.println("      Equals:");
        System.out.println("p.equals(q) = " + p.equals(q));
        System.out.println("p.equals(r) = " + p.equals(r));
    }
}

```

```

        System.out.println("      toString:");
        System.out.println("p.toString() = " + p.toString());
        System.out.println("q.toString() = " + q.toString());
        System.out.println("r.toString() = " + r.toString());
    }
}

Equals:
p.equals(q) = false
p.equals(r) = true
toString:
p.toString() = Shapes.Point@190d11
q.toString() = Shapes.Point@a90653
r.toString() = Shapes.Point@190d11

```

Резултати от тестването без методите clone, equals и toString да са предефинирани:

```

package Shapes;

public class TestPoint {
    public static void main(String[] args) {
        Point p = new Point(-2, 3);
        Point q = new Point(-2, 3);
        Point r = p;
        System.out.println("      Equals:");
        System.out.println("p.equals(q) = " + p.equals(q));
        System.out.println("p.equals(r) = " + p.equals(r));

        System.out.println("      toString:");
        System.out.println("p.toString() = " + p.toString());
        System.out.println("q.toString() = " + q.toString());
        System.out.println("r.toString() = " + r.toString());

        System.out.println("      clone:");
        System.out.println("Point s =(Point)p.clone()");
        Point s = (Point) p.clone();
        System.out.println("s is " + s);
        System.out.println("s == p is " + (s == p));
        System.out.println("s.equals(p) is " + s.equals(p));
    }
}

```

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The method clone() from the type Object is not visible
at Shapes.TestPoint.main([TestPoint.java:17](#))

Предефиниране на методите clone, equals и toString:

```

package Shapes;

public class Point {
    // Data
    private double x, y;

    // Constructor
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // Public methods
    public double getX() {
        return x;
    }
}

```

```

}

public double getY() {
    return y;
}

public double dist(Point p) {
    return Math.sqrt(Math.pow(x - p.x, 2) + Math.pow(y - p.y, 2));
}

// Override
public Object clone() {
    return new Point(x, y);
}

public boolean equals(Object obj) {
    Point p = (Point) obj;
    return p.x == x && p.y == y;
}

public String toString() {
    return "Point(" + x + "," + y + ")";
}
}

```

Резултати от изпълнението:

```

package Shapes;

public class TestPoint {
    public static void main(String[] args) {
        Point p = new Point(-2, 3);
        Point q = new Point(-2, 3);
        Point r = p;
        System.out.println("      Equals:");
        System.out.println("p.equals(q) = " + p.equals(q));
        System.out.println("p.equals(r) = " + p.equals(r));

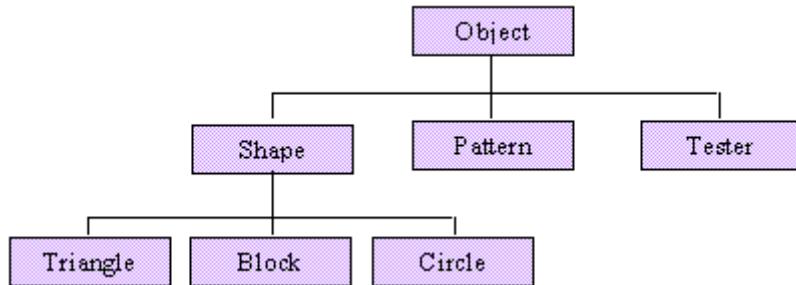
        System.out.println("      toString:");
        System.out.println("p.toString() = " + p.toString());
        System.out.println("q.toString() = " + q.toString());
        System.out.println("r.toString() = " + r.toString());

        System.out.println("      clone:");
        System.out.println("Point s =(Point)p.clone()");
        Point s = (Point) p.clone();
        System.out.println("s is " + s);
        System.out.println("s == p is " + (s == p));
        System.out.println("s.equals(p) is " + s.equals(p));
    }
}

Equals:
p.equals(q) = true
p.equals(r) = true
toString:
p.toString() = Point(-2.0,3.0)
q.toString() = Point(-2.0,3.0)
r.toString() = Point(-2.0,3.0)
clone:
Point s =(Point)p.clone()
s is Point(-2.0,3.0)
s == p is false
s.equals(p) is true

```

Пример: Модификация на разгледаната юерархия от класове:



1. Абстрактен клас **Shape** – реализацията не се променя
2. Модификация на класа **Triangle**: предефиниране на наследените от класа **Object** методи `clone`, `equals` и `toString`

Реализация:

```

package Shapes;

public class Triangle extends Shape {
    // ...

    // Override
    public Object clone() {
        System.out.println("Not implemented!");
        return null;
    }

    public boolean equals(Object obj) {
        System.out.println("Not implemented!");
        return false;
    }

    public String toString() {
        return "Triangle=" + d1.toString() + "," + d2.toString() + ","
            + d3.toString();
    }
}
  
```

3. Модификация на класа **Block**: предефиниране на наследените от класа **Object** методи `clone`, `equals` и `toString`

Реализация:

```

package Shapes;

public class Block extends Shape {
    // ...

    // Override
    public Object clone() {
        return new Block((Point) (d1.clone()), (Point) (d2.clone()));
    }

    public boolean equals(Object obj) {
        Block b = (Block) obj;
        return d1.equals(b.d1) && d2.equals(b.d2);
    }

    public String toString() {
        return "Block=" + d1.toString() + "," + d2.toString();
    }
}
  
```

```
}
```

4. Модификация на класа **Circle**: предефиниране на наследените от класа **Object** методи `clone`, `equals` и `toString`

Реализация:

```
package Shapes;

public class Circle extends Shape {
    // ...

    // Override
    public Object clone() {
        return new Circle((Point) c.clone(), r);
    }

    public boolean equals(Object obj) {
        Circle x = (Circle) obj;
        return r == x.r && c.equals(x.c);
    }

    public String toString() {
        return "Circle=" + c.toString() + "," + r;
    }
}
```

5. Клас **Pattern** за представяне на линеен шаблон от геометрични фигури – реализацията не се променя

6. Реализация на клас **Tester** за тестване

Реализация:

```
package Shapes;

public class Tester {
    private Shape createShape() {
        Shape result = null;
        double x1, y1, x2, y2, x3, y3;
        int kind = (int) (Math.random() * 100) % 3 + 1;
        switch (kind) {
            case 1:
                x1 = Math.random() * 10;
                y1 = Math.random() * 10;
                x2 = Math.random() * 10;
                y2 = Math.random() * 10;
                x3 = Math.random() * 10;
                y3 = Math.random() * 10;
                result = new Triangle(x1, y1, x2, y2, x3, y3);
                break;
            case 2:
                x1 = Math.random() * 10;
                y1 = Math.random() * 10;
                x2 = Math.random() * 10;
                y2 = Math.random() * 10;
                result = new Block(x1, y1, x2, y2);
                break;
            case 3:
                x1 = Math.random() * 10;
                y1 = Math.random() * 10;
                x2 = Math.random() * 10;
                result = new Circle(x1, y1, x2);
        }
    }
}
```

```

        break;
    }
    return result;
}

private void test() {
    System.out.println("      Create pattern");
    int n = (int) (Math.random() * 100) % 10 + 1;
    Pattern p = new Pattern(n);

    for (int i = 0; i < n; i++) {
        Shape temp = createShape();
        p.insert(temp);
        System.out.println("insert: " + temp);
    }

    System.out.println("      Print pattern");
    p.print();

    System.out.println("      Process objects of pattern");
    Point point = new Point(Math.random() * 10, Math.random() * 10);
    for (int i = 0; i < n; i++) {
        Shape temp = p.get(i);
        System.out.println("Shape: " + temp);
        System.out.println("  Area=" + temp.area());
        System.out.println("  P=" + temp.p());
        System.out.println("  " + point + " is member: "
                           + temp.isMember(point));
    }
}

public void start() {
    System.out.println("Start...");
    test();
    System.out.println("Done!");
}

public static void main(String[] args) {
    new Tester().start();
}
}

```

Резултати от изпълнението:

```

Start...
      Create pattern
insert:
Triangle=Point(8.246035793830414,2.11335311610516),Point(7.91070743986805,8.8500
0231486922),Point(2.8435158187537213,8.172946800908994)
insert: Circle=Point(9.909411929687765,5.328277337398949),7.910427540608335
insert: Circle=Point(6.629180939529077,3.1829274634162816),2.446969714097901
      Print pattern
Triangle=Point(8.246035793830414,2.11335311610516),Point(7.91070743986805,8.8500
0231486922),Point(2.8435158187537213,8.172946800908994)
Circle=Point(9.909411929687765,5.328277337398949),7.910427540608335
Circle=Point(6.629180939529077,3.1829274634162816),2.446969714097901
      Process objects of pattern
Shape:
Triangle=Point(8.246035793830414,2.11335311610516),Point(7.91070743986805,8.8500
0231486922),Point(2.8435158187537213,8.172946800908994)
  Area=17.181464142700616
Not implemented!
  P=0.0
Not implemented!
  Point(4.732486376877344,3.3541597214510066) is member: false
Shape: Circle=Point(9.909411929687765,5.328277337398949),7.910427540608335

```

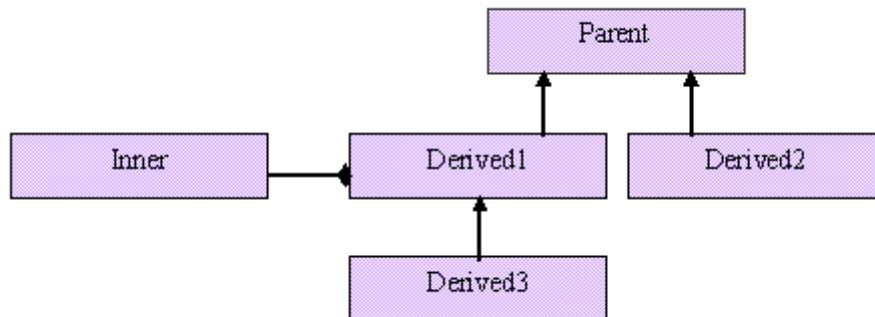
```

Area=196.58473264975626
P=49.70268209665904
Point(4.732486376877344,3.3541597214510066) is member: true
Shape: Circle=Point(6.629180939529077,3.1829274634162816),2.446969714097901
Area=18.810791124015278
P=15.374764154733365
Point(4.732486376877344,3.3541597214510066) is member: true
Done!

```

Следващият пример илюстрира изпълнението на конструкторите по премълчаване на класове, организирани в йерархия, както и обработка на колекция от техни обекти.

Пример: Разглеждаме йерархията от класове, представена чрез диаграмата на наследяване (**Derived1** и **Derived2** са подкласове на **Parent**, **Derived3** е подклас на **Derived1**) и агрегация (класът **Derived1** съдържа променлива от тип **Inner**):



Реализация на класа **Parent**:

```

package OOPExample;

public class Parent {
    // Constructor
    public Parent() {
        System.out.println("Parent: Default constructor");
    }

    // Public method
    public void print() {
        System.out.println("Parent: Print");
    }

    // Public methods toString, equals - inherited from Object
}

```

Parent

Данни

Методи
print
toString
equals

Реализация на класа **Inner**:

```

package OOPExample;

public class Inner {
    // Constructor
    public Inner() {
        System.out.println("Inner: Default constructor");
    }

    // Public method
    public void print() {
        System.out.println("Inner: Print");
    }

    // Public methods toString, equals - inherited from Object
}

```

Inner

Данни

Методи
print
toString
equals

Реализация на класа Derived1:

```

package OOPExample;

public class Derived1 extends Parent {
    // Data
    private Inner obj;

    // Constructor
    public Derived1() {
        System.out.println("Derived1: Default constructor");
        obj = new Inner();
    }

    // Override
    public void print() {
        System.out.println("Derived1: Print");
        super.print();
        obj.print();
    }

    // Public methods toString, equals - inherited from Object
}

```

Derived1

Данни
obj

Методи
print
toString
equals

Реализация на класа Derived3:

```

package OOPExample;

public class Derived3 extends Derived1 {
    // Constructor
    public Derived3() {
        System.out.println("Derived3: Default constructor");
    }

    // Override
    public void print() {
        System.out.println("Derived3: Print");
        super.print();
    }

    // Public methods toString, equals - inherited from Derived1
}

```

Derived3

Данни
obj

Методи
print
toString
equals

Реализация на класа Derived2:

```

package OOPExample;

public class Derived2 extends Parent {
    // Constructor
    public Derived2() {
        System.out.println("Derived2: Default constructor");
    }

    // Override
    public void print() {
        System.out.println("Derived2: Print");
        super.print();
    }

    // Public methods toString, equals - inherited from Object
}

```

Derived2

Данни

Методи
print
toString
equals

Тестване: Реализация на клас Tester

```

package OOPExample;

public class Tester {
    public void start() {
        System.out.println("      Test 1");
        test1();
        System.out.println("      Test 2");
        test2();
    }

    private void test1() {
        System.out.println("Start...");
        System.out.println("      new Derived3()");
        new Derived3();
        System.out.println("Done!");
    }

    private void test2() {
        System.out.println("Start...");
        System.out.println("      Parent p[] = new Parent[4]");
        Parent p[] = new Parent[4];
        System.out.println("      p[0] = new Parent()");
        p[0] = new Parent();
        System.out.println("      p[1] = new Derived1()");
        p[1] = new Derived1();
        System.out.println("      p[2] = new Derived2()");
        p[2] = new Derived2();
        System.out.println("      p[3] = new Derived3()");
        p[3] = new Derived3();
        System.out.println("      Print");
        for (int i = 0; i < 4; i++)
            p[i].print();
        System.out.println("Done!");
    }
}

```

Резултати от тестването:

```

package OOPExample;

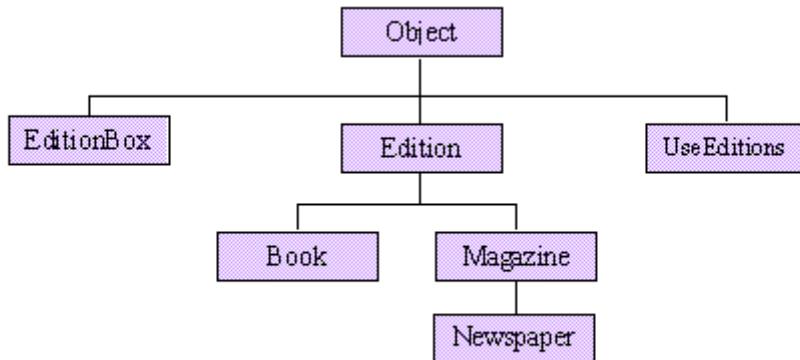
public class Example {
    public static void main(String[] args) {
        System.out.println("      Inheritance And Polymorphism");
        new Tester().start();
    }
}

Inheritance And Polymorphism
Test 1
Start...
      new Derived3()
Parent: Default constructor
Derived1: Default constructor
Inner: Default constructor
Derived3: Default constructor
Done!
      Test 2
Start...
      Parent p[] = new Parent[4]
      p[0] = new Parent()
Parent: Default constructor
      p[1] = new Derived1()
Parent: Default constructor
Derived1: Default constructor
Inner: Default constructor

```

```
p[2] = new Derived2()
Parent: Default constructor
Derived2: Default constructor
p[3] = new Derived3()
Parent: Default constructor
Derived1: Default constructor
Inner: Default constructor
Derived3: Default constructor
    Print
Parent: Print
Derived1: Print
Parent: Print
Inner: Print
Derived2: Print
Parent: Print
Derived3: Print
Derived1: Print
Parent: Print
Inner: Print
Done!
```

Пример: Йерархия от класове:



1. Клас **Edition** за представяне на печатно издание:

Реализация:

```
package Editions;

public class Edition {
    // Data
    private String title;
    private String publish;
    private int year;

    // Constructor
    public Edition(String t, String p, int y) {
        title = t;
        publish = p;
        year = y;
    }

    // Public methods
    public void print() {
        System.out.println("Title: " + title + "\nPublish: " + publish
                           + "\nYear: " + year);
    }

    public int compareTo(Edition arg) {
        return this.year - arg.year;
    }
}
```

Edition

Данни

title

publish

year

Методи

print

compareTo

toString

equals

```
// Override
public String toString() {
    return title + "\n" + publish + "\n" + year + "\n";
}

public boolean equals(Object arg) {
    return toString().equals(((Edition) arg).toString());
}
}
```

2. Клас **Book** за представяне на книга – вид печатно издание:

Реализация:

```
package Editions;

public class Book extends Edition {
    // Data
    // Book has title, publish, year
    private String author;

    // Constructor
    public Book(String t, String p, int y, String a) {
        super(t, p, y);
        author = a;
    }

    // Override
    public void print() {
        super.print();
        System.out.println("Author: " + author);
    }

    public String toString() {
        return super.toString() + author + "\n";
    }

    // Public methods compareTo, equals - inherited from Edition
}
```

Book

Данни
title
publish
year
author

Методи
print
compareTo
toString
equals

3. Клас **Magazine** за представяне на списание – вид печатно издание:

Реализация:

```
package Editions;

public class Magazine extends Edition {
    // Data
    // Magazine has title, publish, year
    private int number;

    // Constructor
    public Magazine(String t, String p, int y, int n) {
        super(t, p, y);
        number = n;
    }

    // // Override
    public void print() {
        super.print();
        System.out.println("Number: " + number);
    }

    public String toString() {
```

Magazine

Данни
title
publish
year
number

Методи
print
compareTo
toString
equals

```

        return super.toString() + number + "\n";
    }

    // Public methods compareTo, equals - inherited from Edition
}

```

4. Клас **Newspaper** за представяне на вестник – вид печатно издание:

Реализация:

```

package Editions;

public class Newspaper extends Magazine {
    // Data
    // Newspaper has title, publish, year, number
    private int day;
    private int month;
    private final static String[] m = { "January", "February",
"March", "April", "May", "June", "July", "August", "September",
"October", "November", "December" };

    // Constructor
    public Newspaper(String t, String p, int y, int n, int d, int m)
    {
        super(t, p, y, n);
        day = d;
        month = m;
    }

    // Override
    public void print() {
        super.print();
        System.out.println("Day: " + day);
        System.out.println("Month: " + m[month - 1]);
    }

    public String toString() {
        return super.toString() + day + "\n" + month + "\n";
    }

    // Public methods compareTo, equals - inherited from Magazine
}

```

Newspaper

Данни

title

publish

year

number

day

month

Mетоди

print

compareTo

toString

equals

5. Клас **EditionBox** за представяне на класър с фиксиран брой места, на всяко от които може да се постави един екземпляр на печатно издание. Местата са номерирани отляво надясно, започвайки от 1. Екземплярите в класъра са разположени на съседни места, започвайки от място с номер 1 и са подредени в нарастващ ред на годината на издаване. Няма повтарящи се екземпляри.

Спецификация:

```

package Editions;

public class EditionBox {
    //Декларации на променливи: array за обектите на класъра, принадлежащи на йерархията
    //с корен Edition и number за броя на обектите

    //Конструктор
    public EditionBox(int n) {...}//Създава празен класър с n места, номерирани с числата от 0 до n-1
    public EditionBox() {...}//Създава празен класър с 20 места
    public EditionBox(Edition[] arg) {...}//Създава класър от елементите на масива arg

    //Собствени методи
}

```

```
private void insertAt(int i, Edition arg) {...}//Включва изданието arg в позиция i на текущия класър
private void removeAt(int i) {...}//Изключва изданието в позиция i на текущия класър
```

```
//Методи
public Edition insert(Edition arg) {...}//Включва изданието arg в сортирания текущ класър, ако има
                                         //свободно място и ако arg не се съдържа в него
public Edition remove(Edition arg) {...}//Изключва изданието arg от текущия класър
public void print() {...}//Извежда текущия класър
}
```

Реализация:

```
package Editions;

public class EditionBox {
    // Data
    private Edition[] array;
    private int number;

    // Constructors
    public EditionBox(int n) {
        number = 0;
        array = new Edition[n];
    }

    public EditionBox() {
        number = 0;
        array = new Edition[20];
    }

    public EditionBox(Edition[] arg) {
        number = arg.length;
        array = new Edition[number];
        for (int k = 0; k < number; k++)
            array[k] = arg[k];
    }

    // Private methods
    private void insertAt(int i, Edition arg) {
        for (int j = number; j > i; j--)
            array[j] = array[j - 1];
        array[i] = arg;
        number++;
    }

    private void removeAt(int i) {
        System.out.println("Not implemented!");
    }

    //Public methods
    public Edition insert(Edition arg) throws RuntimeException {
        if (number == array.length)
            throw new RuntimeException("No space!");
        int i = 0;
        for (; i < number; i++)
            if (arg.compareTo(array[i]) < 0) {
                insertAt(i, arg);
                break;
            } else if (arg.compareTo(array[i]) == 0)
                if (arg.equals(array[i]))
                    break;
            if (i == number)
                insertAt(i, arg);
        return arg;
    }
}
```

```

public Edition remove(Edition arg) {
    System.out.println("Not implemented!");
    return null;
}

public void print() {
    for (int i = 0; i < number; i++) {
        System.out.println();
        array[i].print();
    }
}
}

```

6. Клас **UseEditions** за четене на последователност от издания, съхранение и извеждане:

```

package Editions;

import ccj.*;

public class UseEditions {
    private static Edition read() throws Exception {
        System.out
            .print("Enter the kind: 1-edition,2-book,3-magazine,4-newspaper,0-
end:");
        int c = Integer.parseInt(Console.in.readLine());
        if (c < 1 || c > 4)
            throw new Exception("No more editions!");
        String t = Console.in.readLine();
        String p = Console.in.readLine();
        int y = Integer.parseInt(Console.in.readLine());
        switch (c) {
        case 1:
            return new Edition(t, p, y);
        case 2:
            return new Book(t, p, y, Console.in.readLine());
        case 3:
            int x = Integer.parseInt(Console.in.readLine());
            return new Magazine(t, p, y, x);
        case 4:
            int n = Integer.parseInt(Console.in.readLine());
            int d = Integer.parseInt(Console.in.readLine());
            int m = Integer.parseInt(Console.in.readLine());
            return new Newspaper(t, p, y, n, d, m);
        default:
            return null;
        }
    }

    public static void main(String[] args) {
        System.out.print("Enter the number of places in the box: ");
        int n = Integer.parseInt(Console.in.readLine());
        EditionBox box = new EditionBox(n);
        try {
            while (true)
                box.insert(read());
        } catch (Exception e1) {
            System.out.println(e1.getMessage());
        }

        System.out.println("The contents of the box is:");
        box.print();
    }
}

```

Чрез реализираните до момента класове се дефинират типове данни, които на са част от езика и които се използват подобно на предварително дефинираните типове в него. Следващите задачи илюстрират прилагането на обектно-ориентирания подход за създаване на нови типове данни (*абстрактни типове данни*) и тяхното използване – предмет на следващите теми.

Задача: Да се реализира тип **рационално число** за обработка на рационални числа, предоставящ следните операции:

- Създаване на рационално число
- Унищожаване на рационално число
- Аритметични операции
 - Умножение на рационални числа
 - Деление на рационални числа
 - Събиране на рационални числа
 - Изваждане на рационални числа
- Операции за сравнение на рационални числа ==, !=, <, <=, > и >=
- Операции за вход и изход
 - Четене на рационално число от стандартния вход
 - Извеждане на рационално число на стандартния изход
- Операции за преобразуване
 - Цяло число в рационално число
 - Рационално число в цяло число
 - Рационално число в реално число
 - Рационално число в низ
- Достъп до данните

Обектно-ориентирана реализация. Клас **Ratio** за представяне на рационално число, съгласно спецификацията:

```
package RationalNumbers;

import java.util.Scanner;

public class Ratio {
    //Декларации на променливи за данните на рационално число – числител p и
    //знаменател q>0, като p и q са цели, взаимно прости числа
    private int p;
    private int q;

    //Конструктори за инициализация на променливите
    public Ratio() { p = 0; q = 1; }
    public Ratio(int p) { this.p = p; q = 1; }
    public Ratio(int p,int q) { }

    //Собствен метод
    private static int gcd(int a,int b) { ... }

    //Общодостъпни методи
    //Аритметични операции
    public Ratio add(Ratio r) { ... }
    public Ratio div(Ratio r) { ... }
    public Ratio subtr(Ratio r) { ... }
    public Ratio mult(Ratio r) { ... }

    //Сравнение на рационални числа
    public int compareTo(Ratio r) { ... }

    //Вход/изход
    public static Ratio read() { ... }
```

```

public void print() {...}

//Методи за достъп до данните
public int getNum() {...}
public void setNum(int newP) {...}
public int getDen() {...}
public void setDen(int newQ) {...}

//Методи за преобразуване
public double toDouble() {...}
public intToInt() {...}
public String toString() {...}

//Декларации на константи
public static final Ratio ZERO=new Ratio();
}

```

Реализация:

```

package RationalNumbers;

import java.util.Scanner;

public class Ratio {
    // Data
    private int p; //numerator
    private int q; //denominator

    // Constructors
    public Ratio() {
        p = 0;
        q = 1;
    }

    public Ratio(int p) {
        this.p = p;
        q = 1;
    }

    public Ratio(int p, int q) {
        if (q == 0)
            throw new RuntimeException(
                "RationalNumbers: invalid denominator 0!");
        int a = gcd(p, q);
        this.p = p / a;
        this.q = q / a;
        if (p < 0 && q < 0) {
            this.p = -this.p;
            this.q = -this.q;
        } else if (p < 0 || q < 0) {
            this.p = -Math.abs(this.p);
            this.q = Math.abs(this.q);
        }
    }

    // Private method
    private static int gcd(int a, int b) {
        if (a == 0)
            b = 1;
        else {
            a = Math.abs(a);
            b = Math.abs(b);
            int p = a % b;
            while (p != 0) {
                a = b;

```

```

        b = p;
        p = a % b;
    }
    return b;
}

// Operators
public Ratio add(Ratio r) {
    return new Ratio(this.p * r.q + this.q * r.p, this.q * r.q);
}

public Ratio div(Ratio r) {
    if (r.p == 0)
        throw new ArithmeticException("RationalNumbers: divide by zero!");
    int p = this.p * r.q;
    int q = Math.abs(this.q * r.p);
    return r.p < 0 ? new Ratio(-p, q) : new Ratio(p, q);
}

public Ratio subtr(Ratio r) {
    System.out.println("Unsupported Operation: subtr!");
    return null;
}

public Ratio mult(Ratio r) {
    System.out.println("Unsupported Operation: mult!");
    return null;
}

public int compareTo(Ratio r) {
    return this.p * r.q - this.q * r.p;
}

// Input/output methods
public static Ratio read() {
    Scanner sc = new Scanner(System.in);
    return new Ratio(sc.nextInt(), sc.nextInt());
}

public void print() {
    System.out.println(this.p + "/" + this.q);
}

// Data access methods
public int getNum() {
    return this.p;
}

public void setNum(int newP) {
    int a = gcd(newP, this.q);
    this.p = newP / a;
    this.q = this.q / a;
}

public int getDen() {
    return this.q;
}

public void setDen(int newQ) {
    Ratio r = new Ratio(this.p, newQ);
    this.p = r.p;
    this.q = r.q;
}

// Data conversion methods

```

```

public double toDouble() {
    return (double) this.p / (double) this.q;
}

public int toInt() {
    return this.p / this.q;
}

public String toString() {
    return this.p + "/" + this.q;
}

// Constants
public static final Ratio ZERO = new Ratio();
}

```

Използване на типа Ratio.

- Клас **RatioCalculator** за представяне на калкулатор, който чете двойки от рационални числа и за числата от всяка двойка прилага аритметичните операции и операцията за сравнение и извежда получените резултати

Реализация:

```

package RationalNumbers;

import java.util.Scanner;

public class RatioCalculator {
    // Constructor: public RatioCalculator()

    // Public method
    public void start() {
        Scanner sc = new Scanner(System.in);
        int flag = 1;

        while (flag == 1) {
            try {
                System.out.print("      A = ");
                Ratio A = Ratio.read();
                System.out.print("      B = ");
                Ratio B = Ratio.read();

                System.out.println("A + B = " + A.add(B));
                System.out.println("A - B = " + A.subtr(B));
                System.out.println("A * B = " + A.mult(B));
                System.out.println("A / B = " + A.div(B));
                System.out.println("A < B = " + (A.compareTo(B) < 0));
            } catch (RuntimeException e) {
                System.out.println(e.getMessage());
            }
            System.out.print("Do you want to continue?(1-Yes/0-No): ");
            flag = sc.nextInt();
        }
    }
}

```

- Клас **RatioBubbleSorter** за сортиране на масив от рационални числа по метода на *мехурчето*

Реализация:

```
package RationalNumbers;
```

```

public class RatioBubbleSorter {
    // Constructor: public RatioBubbleSorter()

    // Public methods
    public void sort(Ratio[] array, int from, int to) {
        if (from >= 0 && to >= from && to <= array.length - 1) {
            boolean sorted;
            do {
                sorted = true;
                for (int i = from; i < to; i++) {
                    if (array[i].compareTo(array[i + 1]) > 0) {
                        Ratio temp = array[i];
                        array[i] = array[i + 1];
                        array[i + 1] = temp;
                        sorted = false;
                    }
                }
            } while (!sorted);
        } else
            System.out.println("Unsupported Operation: sort!");
    }

    public void sort(Ratio[] array) {
        sort(array, 0, array.length - 1);
    }
}

```

3. Клас Use RatioBubbleSorter за използване на класа RatioBubbleSorter

Реализация:

```

package RationalNumbers;

import java.util.Scanner;

public class UseRatioBubbleSorter {
    // Data
    public static Ratio numbers[];

    // Methods
    public static void read() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter n > 0: ");
        int n = sc.nextInt();
        numbers = new Ratio[n];
        for (int i = 0; i < n; i++) {
            System.out.print("Enter a ratio: ");
            numbers[i] = Ratio.read();
        }
    }

    public static void print() {
        for (int i = 0; i < numbers.length; i++)
            System.out.println(numbers[i].toString());
    }

    public static void main(String[] args) {
        read();
        new RatioBubbleSorter().sort(numbers);
        System.out.println("\nThe sorted array is: ");
        print();
    }
}

```

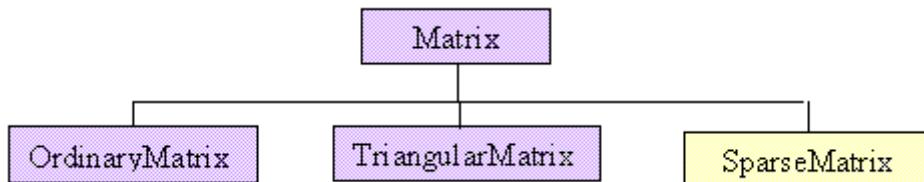
Задача: Да се реализират типове данни за обработка на различни видове **математически матрици**, представени чрез диаграмата:



предоставящи следните операции:

- Създаване и инициализация на матрица
- Получаване на броя на редовете на матрица
- Получаване на броя на стълбовете на матрица
- Получаване на стойност на елемент на матрица по зададени индекси
- Промяна на стойността на елемент на матрица по зададени индекси и нова стойност
- Събиране на матрици
- Умножение на матрици
- Умножение на матрица с число
- Транспониране на матрица
- Проверка дали две матрици са еднакви

Обектно-ориентирана реализация. Реализация на йерархията от класове:



1. Абстрактен клас **Matrix**, съгласно спецификацията:

```

package Matrix;

public abstract class Matrix {
    //Представяне на броя на редовете и стълбовете на матрица
    protected int n; // number of rows
    protected int m; // number of columns

    //Абстрактни методи
    public double elementAt(int i,int j);
    //Връща i,j-ия елемент на текущата матрица
    public void setElementAt(int i,int j,double value);
    //Променя i,j-ия елемент на текущата матрица на value и връща старата стойност
    protected abstract Matrix create(int rows, int cols);
    //Създава и връща нулева матрица с брой редове rows и брой стълбове cols

    protected int f(int i, int j) {...}
    //Проверява дали i и j са коректни индекси на елемент на матрицата. Ако не са, активира изключение
    //IndexOutOfBoundsException. В противен случай връща -1

    //Методи
    public int rows() {...}//Връща броя на редовете на текущата матрица
    public int columns(){...}//Връща броя на стълбовете на текущата матрица
    public Matrix add(Matrix B) {...}//Определя сумата на текущата матрица и матрицата B
    public Matrix mult(Matrix B) {...}//Определя произведението на текущата матрица и матрицата B
    public Matrix mult(double value) {...}//Определя произведенето на value и текущата матрица
    public Matrix transpose() {...}//Определя транспонираната матрица на текущата матрица

    //Предефиниране на наследени методи от класа Object
    public boolean equals(Object obj) {...}//Проверява дали текущата матрица и матрицата obj съвпадат
}
  
```

Реализация:

```

package Matrix;

public abstract class Matrix {
    // Data - number of rows and columns
    protected int n; // number of rows
    protected int m; // number of columns

    // Abstract methods
    public abstract double elementAt(int i, int j);

    public abstract double setElementAt(int i, int j, double value);

    protected abstract Matrix create(int rows, int cols);

    // Protected method
    protected int f(int i, int j) {
        if (i < 1 || i > n || j < 1 || j > m)
            throw new IndexOutOfBoundsException("Incorrect indexes!");
        return -1;
    }

    // Public methods
    public int getRows() {
        return n;
    }

    public int getColumns() {
        return m;
    }

    public Matrix add(Matrix B) {
        Matrix A = this;
        if (B.n != A.n || B.m != A.m)
            throw new RuntimeException("Illegal matrix dimensions!");
        Matrix C = create(n, m);
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
                C.setElementAt(i, j, A.elementAt(i, j) + B.elementAt(i, j));
        return C;
    }

    public Matrix mult(Matrix B) {
        System.out.println("Not implemented!");
        return null;
    }

    public Matrix mult(double value) {
        System.out.println("Not implemented!");
        return null;
    }

    public Matrix transpose() {
        System.out.println("Not implemented!");
        return null;
    }

    public boolean equals(Object obj) {
        System.out.println("Not implemented!");
        return false;
    }
}

```

2. Клас **OrdinaryMatrix** за представяне на матрица, съгласно спецификацията:

```

package Matrix;

public class OrdinaryMatrix extends Matrix {
    //Представяне на елементите на матрица

    //Конструктори
    public OrdinaryMatrix(int rows, int cols) {...}
    //Създава нулева матрица с брой редове rows и брой стълбове cols
    public OrdinaryMatrix(double[][] data) {...}
    //Създава матрица от елементите на масива data
    public OrdinaryMatrix(Matrix arg) {...}
    //Създава матрица-копие на матрицата arg

    //Реализация на абстрактните методи на класа Matrix
    public double elementAt(int i, int j) {...}
    public void setElementAt(int i, int j, double value) {...}
    protected Matrix create(int rows, int cols)

    //Наследяване на методите, реализирани в класа Matrix
}

```

Следват два варианта на реализация на класа **OrdinaryMatrix**.

Вариант 1: Реализацията на класа **OrdinaryMatrix** са основава на представянето на елементите a_{ij} на матрица $A(n \times m)$, $i \in [1; n]$, $j \in [1; m]$ чрез масив **double[][] elements**, като $\text{elements}[i-1][j-1] = a_{ij}$

```

package Matrix;

public class OrdinaryMatrix extends Matrix {
    // Data-elements
    private double[][] elements;

    // Constructors
    public OrdinaryMatrix(int rows, int cols) {
        n = rows;
        m = cols;
        elements = new double[n][m];
    }

    public OrdinaryMatrix(double[][] data) {
        this(data.length, data[0].length);
        for (int i = 0; i < data.length; i++)
            for (int j = 0; j < data[0].length; j++)
                elements[i][j] = data[i][j];
    }

    public OrdinaryMatrix(Matrix arg) {
        this(arg.getRows(), arg.getColumns());
        for (int i = 1; i <= arg.getRows(); i++)
            for (int j = 1; j <= arg.getColumns(); j++)
                elements[i - 1][j - 1] = arg.elementAt(i, j);
    }

    // Protected method f - inherited from Matrix

    // Implementation of abstract methods
    public double elementAt(int i, int j) {
        f(i, j);
        return elements[i - 1][j - 1];
    }

    public void setElementAt(int i, int j, double value) {
        f(i, j);
        elements[i - 1][j - 1] = value;
    }
}

```

```

        double oldValue = elements[i - 1][j - 1];
        elements[i - 1][j - 1] = value;
        return oldValue;
    }

    protected Matrix create(int rows, int cols) {
        return new OrdinaryMatrix(rows, cols);
    }

    // Public methods - inherited from Matrix
}

```

Вариант 2: Реализацията на класа *OrdinaryMatrix* се основава на представянето на елементите a_{ij} на матрица $A(n \times m)$, $i \in [1; n]$, $j \in [1; m]$ чрез масив **double[] elements** с дължина $n \times m$, като $\text{elements} = (a_{11}, \dots, a_{1m}, a_{21}, \dots, a_{2m}, \dots, a_{n1}, \dots, a_{nm})$ и $\text{elements}[(i-1)*m+j-1] = a_{ij}$

```

package Matrix;

public class OrdinaryMatrix extends Matrix {
    // Data-elements
    private double[] elements;

    // Constructors
    public OrdinaryMatrix(int rows, int cols) {
        n = rows;
        m = cols;
        elements = new double[n * m];
    }

    public OrdinaryMatrix(double[][] data) {
        this(data.length, data[0].length);
        for (int i = 0; i < data.length; i++)
            for (int j = 0; j < data[0].length; j++)
                setElementAt(i + 1, j + 1, data[i][j]);
    }

    public OrdinaryMatrix(Matrix arg) {
        this(arg.getRows(), arg.getColumns());
        for (int i = 1; i <= arg.getRows(); i++)
            for (int j = 1; j <= arg.getColumns(); j++)
                setElementAt(i, j, arg.elementAt(i, j));
    }

    // Override
    protected int f(int i, int j) {
        super.f(i, j);
        return (i - 1) * m + j - 1;
    }

    // Implementation of abstract methods
    public double elementAt(int i, int j) {
        return elements[f(i, j)];
    }

    public double setElementAt(int i, int j, double value) {
        int k = f(i, j);
        double oldValue = elements[k];
        elements[k] = value;
        return oldValue;
    }

    protected Matrix create(int rows, int cols) {
        return new OrdinaryMatrix(rows, cols);
    }
}

```

```
// Public methods - inherited from Matrix
}
```

3. Клас **TriangularMatrix** за представяне на триъгълна матрица – матрица, в която елементите над главния диагонал са нули, съгласно спецификацията:

```
package Matrix;
```

```
public class TriangularMatrix extends Matrix {
    //Представяне на елементите на матрица

    //Конструктори
    public TriangularMatrix(int rows, int cols) {...}
    //Създава нулева матрица с брой редове rows и брой стълбове cols, ако rows = cols.
    //В противен случай активира изключение RuntimeException
    public TriangularMatrix(double[][] data) {...}
    //Създава матрица от елементите на масива data
    public TriangularMatrix(Matrix arg) {...}
    //Създава матрица-копие на матрицата arg

    //Реализация на абстрактните методи на класа Matrix
    public double elementAt(int i, int j) {...}
    public double setElementAt(int i, int j, double value) {...}
    protected Matrix create(int rows, int cols)

    //Наследяване на методите, реализирани в класа Matrix
    //Предефиниране на наследения метод f от класа Matrix
    //Предефиниране на наследения метод add от класа Matrix
    //Собствен метод plus за събиране на триъгълни матрици
}
```

Реализацията на класа **TriangularMatrix** са основава на представянето на елементите a_{ij} под и по главния диагонал на триъгълна матрица $A(n \times m)$, $n=m$, $i \in [1; n]$, $j \in [1; i]$ чрез масив **double[] elements** с дължина $((1+n)*n)/2$, като $\text{elements}=(a_{11}, a_{21}, a_{22}, \dots, a_{n1}, \dots, a_{nn})$ и $\text{elements}[(i*(i-1))/2+j-1]=a_{ij}$

```
package Matrix;

public class TriangularMatrix extends Matrix {
    // Data-elements
    private double[] elements;

    // Constructors
    public TriangularMatrix(int rows, int cols) {
        if (rows != cols)
            throw new RuntimeException("Illegal matrix dimensions!");
        n = m = rows;
        elements = new double[((1 + n) * n) / 2];
    }

    public TriangularMatrix(double[][] data) {
        this(data.length, data[0].length);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                setElementAt(i + 1, j + 1, data[i][j]);
    }

    public TriangularMatrix(Matrix arg) {
        this(arg.getRows(), arg.getColumns());
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                setElementAt(i, j, arg.elementAt(i, j));
    }
}
```

```

// Override
protected int f(int i, int j) {
    super.f(i, j);
    return (i * (i - 1)) / 2 + j - 1;
}

// Implementation of abstract methods
public double elementAt(int i, int j) {
    int k = f(i, j);
    double result = elements[k];
    return j <= i ? result : 0;
}

public double setElementAt(int i, int j, double value) {
    int k = f(i, j);
    double oldValue = elements[k];
    if (j <= i)
        elements[k] = value;
    else if (value != 0)
        throw new RuntimeException("Incorrect value!");
    return oldValue;
}

protected Matrix create(int rows, int cols) {
    return new TriangularMatrix(rows, cols);
}

// Override
public Matrix add(Matrix B) {
    if (B instanceof TriangularMatrix)
        return plus((TriangularMatrix) B);
    Matrix A = new OrdinaryMatrix(this);
    return A.add(B);
}

// Private method
private TriangularMatrix plus(TriangularMatrix B) {
    Matrix A = this;
    if (B.n != A.n)
        throw new RuntimeException("Illegal matrix dimensions!");
    TriangularMatrix C = new TriangularMatrix(n, n);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++)
            C.setElementAt(i, j, A.elementAt(i, j) + B.elementAt(i, j));
    return C;
}

// Other public methods - inherited from Matrix
}

```

4. Клас **SparseMatrix** за представяне на разредена матрица – матрица, в която повечето елементи са нули, съгласно спецификацията:

```

package Matrix;

public class SparseMatrix extends Matrix {
    //Представяне на елементите на матрица

    //Конструтори
    public SparseMatrix(int rows, int cols) {...}
    //Създава нулева матрица с брой редове rows и брой стълбове cols
    public SparseMatrix(double[][] data) {...}
    //Създава матрица от елементите на масива data
    public SparseMatrix(Matrix arg) {...}
    //Създава матрица-копие на матрицата arg
}

```

```
//Реализация на абстрактните методи на класа Matrix
public double elementAt(int i, int j) {...}
public double setElementAt(int i, int j, double value) {...}
protected Matrix create(int rows, int cols)

//Наследяване на методите, реализирани в класа Matrix
//Предефиниране на някои от наследените методи на класа Matrix, ако е необходимо
}
```

Реализацията на класа **SparseMatrix** се основава на представянето на множеството $\text{elements}=\{(i,j,a_{ij}) \mid a_{ij}\neq 0\}$ от ненулевите елементи a_{ij} на матрица $A(n \times m)$, $i \in [1; n]$, $j \in [1; m]$.

Използване. Клас **TestMatrix** за тестване:

```
package Matrix;

public class TestMatrix {
    public static void print(Matrix A) {
        for (int i = 1; i <= A.getRows(); i++)
            for (int j = 1; j <= A.getColumns(); j++)
                System.out.println("El[" + i + "," + j + "] = " +
                    + A.elementAt(i, j));
    }

    public static void main(String[] args) {
        double[][] d = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
        Matrix M = new OrdinaryMatrix(d);
        System.out.println("      OrdinaryMatrix M:");
        print(M);

        double[][] c = { { 9, 0, 0 }, { 6, 5, 0 }, { 3, 2, 1 } };
        Matrix T = new TriangularMatrix(c);
        System.out.println("      TriangularMatrix T:");
        print(T);

        System.out.println("      Matrix M+M:");
        print(M.add(M));

        System.out.println("      Matrix M+T:");
        print(M.add(T));

        System.out.println("      Matrix T+T:");
        print(T.add(T));

        System.out.println("      Matrix T+M:");
        print(T.add(M));
    }
}
```

Резултати от тестването:

```
OrdinaryMatrix M:
El[1,1] = 1.0
El[1,2] = 2.0
El[1,3] = 3.0
El[2,1] = 4.0
El[2,2] = 5.0
El[2,3] = 6.0
El[3,1] = 7.0
El[3,2] = 8.0
El[3,3] = 9.0
TriangularMatrix T:
El[1,1] = 9.0
```

```
E1[1,2] = 0.0
E1[1,3] = 0.0
E1[2,1] = 6.0
E1[2,2] = 5.0
E1[2,3] = 0.0
E1[3,1] = 3.0
E1[3,2] = 2.0
E1[3,3] = 1.0
    Matrix M+M:
E1[1,1] = 2.0
E1[1,2] = 4.0
E1[1,3] = 6.0
E1[2,1] = 8.0
E1[2,2] = 10.0
E1[2,3] = 12.0
E1[3,1] = 14.0
E1[3,2] = 16.0
E1[3,3] = 18.0
    Matrix M+T:
E1[1,1] = 10.0
E1[1,2] = 2.0
E1[1,3] = 3.0
E1[2,1] = 10.0
E1[2,2] = 10.0
E1[2,3] = 6.0
E1[3,1] = 10.0
E1[3,2] = 10.0
E1[3,3] = 10.0
    Matrix T+T:
E1[1,1] = 18.0
E1[1,2] = 0.0
E1[1,3] = 0.0
E1[2,1] = 12.0
E1[2,2] = 10.0
E1[2,3] = 0.0
E1[3,1] = 6.0
E1[3,2] = 4.0
E1[3,3] = 2.0
    Matrix T+M:
E1[1,1] = 10.0
E1[1,2] = 2.0
E1[1,3] = 3.0
E1[2,1] = 10.0
E1[2,2] = 10.0
E1[2,3] = 6.0
E1[3,1] = 10.0
E1[3,2] = 10.0
E1[3,3] = 10.0
```



Задача: Да се реализира клас **IntegersPrinter** за генериране на случаен брой от случаини цели числа и извеждане на числата в ред обратен на получаването им, като се използва родовият стек **java.util.Stack<E>** - виж <http://java.sun.com/javase/6/docs/api/>.

Реализация:

```
package Stack;

import java.util.Stack;

public class IntegersPrinter {
    public void print() {
        Stack<Integer> integerStack = new Stack<Integer>();
        System.out.println("Starting...");

        int n = (int) (Math.random() * 100) % 10 + 1;
        int number;

        for (int i = 0; i < n; i++) {
            number = (int) (Math.random() * 100);
            integerStack.push(number);
            System.out.println("push: " + number);
        }

        while (!integerStack.isEmpty()) {
            number = integerStack.pop();
            System.out.println("pop: " + number);
        }

        System.out.println("Done!");
    }
}

class TestIntegersPrinter {
    public static void main(String[] args) {
        (new IntegersPrinter()).print();
    }
}
```

Задача: Да се реализира клас **TextPrinter** за въвеждане на текст и извеждане на последните n (случайно число) реда от него, като се използва родовият стек **java.util.Stack<E>**.

Реализация:

```
package Stack;

import java.util.Stack;
import java.util.Scanner;

public class TextPrinter {
    public void print() {
        Stack<String> stringStack = new Stack<String>();
        Scanner s = new Scanner(System.in);

        int n = (int) (Math.random() * 100) % 10 + 1;
        System.out.println("Reads text and prints last " + n
                           + " (or less) lines:");

        System.out.println("Reading...");
    }
}
```

```

String line = s.nextLine();
while (!line.equals("")) {
    stringStack.push(line);
    line = s.nextLine();
}

String result = "";
for (int i = 1; i <= n; i++)
    if (!stringStack.isEmpty())
        result = stringStack.pop() + '\n' + result;
    else
        break;
System.out.println("Printing last lines (" + n + " or less):");
System.out.println(result);

System.out.println("Done!");
}
}

class TestTextPrinter {
    public static void main(String[] args) {
        (new TextPrinter()).print();
    }
}

```

Задача: Да се реализира стек с елементи от тип **Тип** с ограничен капацитет, съгласно спецификацията:

```

package Stack;

import java.util.EmptyStackException;

public class ИмеНаСтек {
    //Представяне на стек чрез масив
    private Тип[] items;
    private int top;

    //Конструктори
    public ИмеНаСтек () {...} //Създава празен стек
    public ИмеНаСтек (int n) {...} //Създава празен стек с капацитет n

    //Методи
    public boolean isEmpty() {...} //Проверява дали текущия стек е празен
    public boolean isFull() {...} //Проверява дали текущия стек е пълен
    public void push(Тип x) throws FullStackException {...} //Включва x в текущия стек
    public Тип pop() throws EmptyStackException {...} //Изключва елемента на върха на текущия стек
    public Тип top() throws EmptyStackException {...} //Връща елемента на върха на текущия стек
    public int size() {...} //Връща броя на елементите на текущия стек
    public int capacity() {...} //Връща капацитета на текущия стек
    public void clear() {...} //Променя текущия стек на празен
}

```

Където:

Вариант 1: Тип = **char** и ИмеНаСтек = **StackOfChars**

Вариант 2: Тип = **int** и ИмеНаСтек = **StackOfIntegers**

Реализация на собствено изключение:

```

package Stack;

@SuppressWarnings("serial")
public class FullStackException extends RuntimeException {
    public FullStackException(String s) {

```

```

        super(s);
    }
}

```

Задача: Да се реализира клас **PalindromChecker** за проверка дали дума принадлежи на езика $L=\{\omega @ O(\omega) \mid \omega \in \{a,b\}^*\}$.

Реализация:

```

package Stack;

import java.util.Scanner;

public class PalindromChecker {
    public void check(String str) {
        StackOfChars stack = new StackOfChars(str.length());

        int step = 1;
        boolean flag = true;

        for (int i = 0; i < str.length(); i++) {
            char s = str.charAt(i);
            if (step == 1) {
                if (s == 'a' || s == 'b')
                    stack.push(s);
                else if (s == '@')
                    step = 2;
                else {
                    System.out.println("Error: " + s + " at " + (i + 1));
                    flag = false;
                    break;
                }
            } // if
            else { // step = 2
                if (stack.isEmpty())
                    System.out.println("Error: missing symbols before @");
                flag = false;
                break;
            } else if (s != 'a' && s != 'b') {
                System.out.println("Error: " + s + " at " + (i + 1));
                flag = false;
                break;
            } else {
                char c = stack.pop();
                if (c != s) {
                    System.out.println("Error: " + s + " at " + (i + 1));
                    flag = false;
                    break;
                }
            }
        } // else
    } // for

    if (flag && step == 1) {
        System.out.println("Error: missing symbol @");
        flag = false;
    }

    if (flag) {
        if (!stack.isEmpty())
            System.out.println("Error: missing symbols after @");
        else
            System.out.println("Yes");
    }
} // method

```

```

} // class

class TestPalindromChecker {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("To stop testing enter: No more");
        System.out.println("Starting...");
        while (true) {
            System.out.print("      Word: ");
            String word = s.nextLine();
            if (word.equals("No more")) {
                System.out.println("End of test!");
                break;
            } else
                (new PalindromChecker()).check(word);
        }
    }
}

```

Задача: Да се реализира клас **AkermanEvaluator** за итеративно пресмятане на стойността на функцията на Акерман за дадени стойности на променливите, като се използва стек с елементи от тип **int**. Дефиницията на функцията на Акерман е:

$$A(m,n) = \begin{cases} n+1, & \text{ако } m=0 \\ A(m-1,1), & \text{ако } m \neq 0 \text{ и } n=0 \\ A(m-1,A(m,n-1)), & \text{ако } m \neq 0 \text{ и } n \neq 0 \end{cases}$$

За някои стойности на m има формули, които може да се използват при тестването:

$$\begin{array}{ll} A(0,n)=n+1 & A(3,n)=2^{n+3}-3 \\ A(1,n)=n+2 & \\ A(2,n)=2n+3 & A(4,n)=2^2-3, \text{ } n+2 \text{ вложени степени} \end{array}$$

Реализация:

```

package Stack;

import java.util.EmptyStackException;
import java.util.Scanner;

public class AkermanEvaluator {
    public int evaluate(int m, int n) {
        StackOfIntegers intStack = new StackOfIntegers();
        intStack.push(m);
        intStack.push(n);
        while (true) {
            try {
                n = intStack.pop();
                m = intStack.pop();
                if (m != 0) {
                    if (n != 0) {
                        intStack.push(m - 1);
                        intStack.push(m);
                        intStack.push(n - 1);
                    } else {
                        intStack.push(m - 1);
                        intStack.push(1);
                    }
                } else
                    intStack.push(n + 1);
            } // try
        }
    }
}

```

```
        catch (EmptyStackException e) {
            return n;
        } catch (FullStackException e) {
            throw new ArithmeticException("Impossible computing the value!");
        }
    } // while
} // method
} // class

class TestAkermanEvaluator {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("To stop testing enter: m = 0, n = 0");
        System.out.println("Starting...");
        while (true) {
            try {
                System.out.print("      m = ");
                int m = s.nextInt();
                System.out.print("      n = ");
                int n = s.nextInt();
                if (m == 0 && n == 0) {
                    System.out.println("End of test!");
                    break;
                }
                AkermanEvaluator w = new AkermanEvaluator();
                System.out.println("A(" + m + ", " + n + ") = "
                    + w.evaluate(m, n));
            } catch (ArithmeticalException e) {
                System.out.println(e.getMessage());
            }
        }
    }
}
```

Задача: Да се реализира клас **PostfixExpressionEvaluator** за пресмятане на стойността на аритметичен израз в обратен полски запис с операнди – цели числа без знак и операции: +, -, *, /:

Решение: Следващата таблица съдържа аритметични изрази в *инфиксен запис* и съответното им представяне в *обратен полски запис*:

Инфиксен запис	Обратен полски запис
$2 + (15 - 12) * 17$	2 15 12 -17 *+
B	B
A+B	A B +
A+B+C	A B +C +
A+B*C	A B C *+
A*(B+C)	A B C +*
A/B*C	A B /C *

Алгоритъм за пресмятане на стойността на аритметичен израз в обратен полски запис:

- ## 1. Създаване на празен стек

За всеки елемент на изра

• елементът е число:

三三

Че:

ментът е операция:

Четене и изключване на десния операнд

и изключване на левия operand от стека

Включване на резултата в стека

Иначе:

Има грешка в израза

3. Четене и изключване на число от стека

4. Проверка:

Ако стекът не е празен:

Има грешка в израза

Иначе:

Последното прочетено число е стойността на израза

Пример: Пресмятане на стойността на $2 \cdot 15 - 12 + 3 \cdot 17$:

2

2, 15

2, 15, 12

2, 3 //3=15-12

2, 3, 17

2, 51 //51=3*17

53 //53=2+51

Реализация:

```
package Stack;

import java.util.EmptyStackException;
import java.util.Scanner;

public class PostfixExpressionEvaluator {
    // Data for all objects
    private static final String operators = "+-*/";

    // Private method
    private static boolean isOperator(char arg) {
        return operators.indexOf(arg) != -1;
    }

    // Public method
    public int evaluate(String str) throws ArithmeticException {
        StackOfIntegers stack = new StackOfIntegers(str.length());

        for (int i = 0; i < str.length(); i++) {
            try {
                char c = str.charAt(i);
                int number = -1;
                if (c >= '0' && c <= '9') {
                    number = 0;
                    while (c != ' ') {
                        number = number * 10 + (c - '0');
                        i++;
                        c = str.charAt(i);
                    }
                    stack.push(number);
                    System.out.println("Pushed constant " + number);
                } else if (isOperator(c)) {
                    int x, y, answer = 0;
                    y = stack.pop();
                    System.out.println("Popped the right operand " + y
                        + " of the " + c);
                    x = stack.pop();
                    System.out.println("Popped the left operand " + x
                        + " of the " + c);
                    switch (c) {
                        case '+':
                            answer = x + y;
                            break;
                        case '-':
                            answer = x - y;
                            break;
                        case '*':
                            answer = x * y;
                            break;
                        case '/':
                            answer = x / y;
                            break;
                    }
                    stack.push(answer);
                }
            } catch (EmptyStackException e) {
                System.out.println("Stack underflow");
                throw new ArithmeticException("Stack underflow");
            }
        }
        return stack.pop();
    }
}
```

```

        case '+':
            answer = x + y;
            break;
        case '-':
            answer = x - y;
            break;
        case '*':
            answer = x * y;
            break;
        case '/':
            answer = x / y;
            break;
    }
    stack.push(answer);
    System.out.println("Evaluated " + c + " and pushed "
        + answer);
} else
    throw new ArithmeticException("Invalid operator found: "
        + c);
} catch (EmptyStackException e) {
    throw new ArithmeticException(
        "Not enough numbers in expression!");
}
} // for

if (stack.isEmpty())
    throw new ArithmeticException("No expression provided!");
int value = stack.pop();
System.out.println("Popped " + value + " at end of expression.");
if (!stack.isEmpty())
    throw new ArithmeticException(
        "Not enough operators for all the numbers!");
return value;
} // method
} // class

class TestPostfixExpressionEvaluator {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("To stop testing enter: No more");
        System.out.println("Starting...");
        while (true) {
            System.out.print("      Expression: ");
            String expression = s.nextLine();
            if (expression.equals("No more")) {
                System.out.println("End of test!");
                break;
            } else {
                try {
                    PostfixExpressionEvaluator w = new
PostfixExpressionEvaluator();
                    System.out.println("Value = " + w.evaluate(expression));
                } catch (ArithmeticException e) {
                    System.out.println(e.getMessage());
                }
            }
        }
    }
}

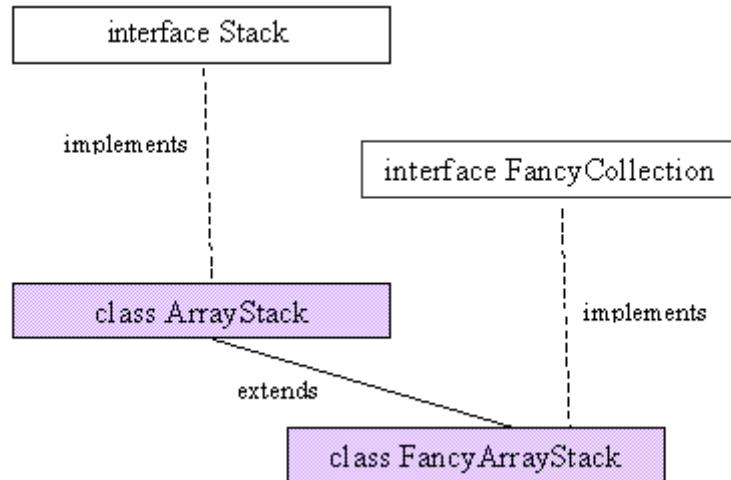
```

Тестване: Изпълнение на теста за следните изрази:

1 2 +7 +
10 5 2 3 +/
1 2 +++
1 2 3 4 5 ++

```
1 2 &
10 2 15 3 12 +/-
2 15 12 -17 *-
1 2 3 4 5 6 7 ++++++
12 0 /
```

Задача: Дадена е йерархията от интерфейси и класове:



Интерфейсът **FancyCollection** представя операции на колекция с елементи от тип **Object**:

```

package Interface;

public interface FancyCollection {
    public boolean isEmpty();
    //Проверява дали текущата колекция и колекцията arg съвпадат

    public int size();
    //Връща броя на елементите на текущата колекция

    public void clear();
    //Променя състоянието на текущата колекция на празна

    public boolean contains(Object arg);
    //Проверява дали arg принадлежи на текущата колекция

    public Object[] toArray();
    //Връща масив от елементите на текущата колекция

    public Object clone();
    //Създава и връща копие на текущата колекция

    public boolean equals(Object arg);
    //Проверява дали текущата колекция и колекцията arg съвпадат

    public String toString();
    //Създава и връща низ от елементите на текущата колекция
}
```

Интерфейсът **Stack** представя операции на стек с елементи от тип **Object**:

```

package Interface;

public interface Stack {
    public boolean isEmpty();
    //Проверява дали текущия стек е пълен

    public void push(Object x);
    //Включва x на върха на текущия стек
  
```

```

public Object pop();
//Изключва елемента на върха на текущия стек и го връща

public Object top();
//Връща елемента на върха на текущия стек без да го изключва
}

```

Да се реализират:

- Клас **ArrayStack** - стек с елементи от тип **Object** с ограничен капацитет, съгласно спецификацията:

```

package Stack;

import java.util.EmptyStackException;
import Interface.Stack;

public class ArrayStack implements Stack {
    //Представяне на стек чрез масив
    protected Object[] items;
    protected int top;

    //Конструктори
    public ArrayStack() {...} //Създава празен стек
    public ArrayStack(int n) {...} //Създава празен стек с капацитет n
    public ArrayStack(Object[] arg) {...} //Създава стек от елементите на масива arg, като първият
        //елемент на масива е на дъното на стека, а последният елемент – на върха
    //Реализация на методите на интерфейса Stack, така че: 1) методите pop и top активират изключение
    //java.util.EmptyStackException, ако текущият стек е празен и 2) методът push активира собствено
    //изключение FullStackException, ако текущият стек е пълен

    //Други методи
    public boolean isFull() {...} //Проверява дали текущия стек е пълен
    public int capacity() {...} //Връща капацитета на текущия стек
}

```

Реализация:

```

package Stack;

import java.util.EmptyStackException;
import Interface.Stack;

public class ArrayStack implements Stack {
    // Data
    protected Object[] items;
    protected int top;

    // Constructors
    public ArrayStack() {
        this(10);
    }

    public ArrayStack(int n) {
        items = new Object[n];
        top = -1;
    }

    public ArrayStack(Object[] data) {
        this(data.length + 10);
        for (int i = 0; i < data.length; i++)
            push(data[i]);
    }
}

```

```
// Stack methods implementation
public boolean isEmpty() {
    return top == -1;
}

public void push(Object x) throws FullStackException {
    if (isFull())
        throw new FullStackException("Stack is full!");
    items[++top] = x;
}

public Object pop() throws EmptyStackException {
    Object result = top();
    top--;
    return result;
}

public Object top() throws EmptyStackException {
    if (isEmpty())
        throw new EmptyStackException();
    return items[top];
}

// Other methods
public boolean isFull() {
    return top == items.length - 1;
}

public int capacity() {
    return items.length;
}
}
```

2. Клас FancyArrayStack, съгласно спецификацията:

```
package Stack;

import Interface.FancyCollection;

public class FancyArrayStack extends ArrayStack implements FancyCollection {
    //Конструтори
    public FancyArrayStack() {...} //Създава празен стек
    public FancyArrayStack(int n) {...} //Създава празен стек с капацитет n
    public FancyArrayStack(FancyCollection arg) {...} //Създава стек от елементите на колекцията arg
    public FancyArrayStack(Object[] arg) {...} //Създава стек от елементите на масива arg, като първият
                                                //елемент на масива е на дъното на стека, а последният елемент – на върха
    //Наследяване на методите на класа ArrayStack
    //Реализация на методите на интерфейса FancyCollection
    //Предефиниране на методите, наследени от клас Object
}
```

Реализация:

```
package Stack;

import Interface.FancyCollection;

public class FancyArrayStack extends ArrayStack implements FancyCollection {
    // Constructors
    public FancyArrayStack() {
        super();
    }

    public FancyArrayStack(int n) {
        super(n);
    }
}
```

```

}

public FancyArrayStack(Object[] data) {
    super(data);
}

public FancyArrayStack(FancyCollection arg) {
    this(arg.toArray());
}

// Methods - inherited from class ArrayStack

// FancyCollection methods implementation
public int size() {
    return top + 1;
}

public void clear() {
    top = -1;
}

public boolean contains(Object arg) {
    for (int i = 0; i <= top; i++)
        if (this.items[i].equals(arg))
            return true;
    return false;
}

public Object[] toArray() {
    Object[] result = new Object[top + 1];
    for (int i = 0; i <= top; i++)
        result[i] = items[i];
    return result;
}

public Object clone() {
    return new FancyArrayStack(this);
}

// Override methods - inherited from class Object
public boolean equals(Object arg) {
    FancyArrayStack s = (FancyArrayStack) arg;
    if (this.size() != s.size())
        return false;
    // Proverka dali masivite this.items i s.items savpadat
    for (int i = 0; i <= top; i++)
        if (!this.items[i].equals(s.items[i]))
            return false;
    return true;
}

public String toString() {
    // Polu4avane na niz ot elementite na masiva items
    String result = "";
    for (int i = 0; i <= top; i++)
        result = items[i] + "\n" + result;
    return result;
}
}

```

3. Клас **PalindromChecker2** за проверка дали дума е палиндром, като се използва класа **FancyArrayStack**.

Реализация:

```

package Stack;

import java.util.Scanner;

public class PalindromChecker2 {
    public void check(String str) {
        FancyArrayStack word = new FancyArrayStack(str.length());
        for (int i = 0; i < str.length(); i++)
            word.push(str.charAt(i));

        FancyArrayStack wordCopy = (FancyArrayStack) word.clone();
        FancyArrayStack reverseWord = new FancyArrayStack(str.length());
        while (!wordCopy.isEmpty())
            reverseWord.push(wordCopy.pop());

        System.out.println(word.equals(reverseWord) ? "Yes" : "No");
    }
}

class TestPalindromChecker2 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("To stop testing enter: No more");
        System.out.println("Starting...");
        while (true) {
            System.out.print("      Word: ");
            String word = s.nextLine();
            if (word.equals("No more")) {
                System.out.println("End of test!");
                break;
            } else
                (new PalindromChecker2()).check(word);
        }
    }
}

```



Задачи за упражнение

Задача: Да се реализира клас **StringReverser** за получаване на обратния низ на даден низ.

Задача: Да се реализира клас **BracketChecker** за проверка дали в даден низ са използвани правилно лява и дясна скоби (), [], {}.

Задача: Да се реализира клас за проверка дали дума принадлежи на езика $L=\{a^m b^{2m+1} \mid m \geq 0\}$.

Задача: Да се реализира клас за проверка дали дума принадлежи на езика $L=\{a^m b^n \mid m > n \geq 0\}$.

Задача: Да се реализира клас **PostfixToInfixConvertor** за преобразуване на аритметичен израз в обратен полски запис с операнди - цели числа без знак и операции: +, -, *, / в аритметичен израз в инфиксен запис.

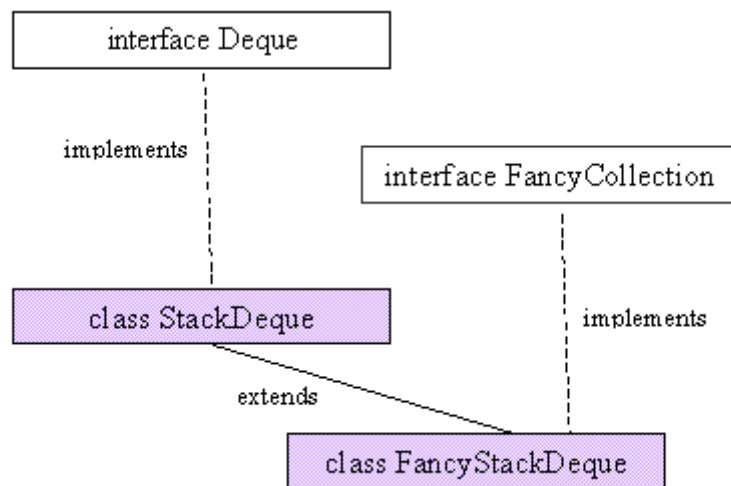
Упътване: Използвайте алгоритъма за пресмятане на стойността на аритметичен израз в обратен полски запис, както е показано на примера:

Пример: Преобразуване на $2 \ 15 \ 12 \ -17 \ *+$ в $(2+((15-12)*17))$.

Задача: Да се реализира клас **InfixToPostfixConvertor** за преобразуване на аритметичен израз в инфиксен запис с операнди - цели числа без знак и операции: +, -, *, / в аритметичен израз в обратен полски запис.



Задача: Дадена е юерархията:



Интерфейсът **Deque** представя операции на *дек с елементи от тип Object*:

```

package Interface;

public interface Deque {
    public boolean isEmpty();
    //Проверява дали текущия дек е празен

    public void addFront(Object x);
    //Включва x в началото (левия край) на текущия дек

    public void addRear(Object x);
    //Включва x накрая (десния край) на текущия дек

    public Object removeFront();
    //Изключва елемента в началото на текущия дек и го връща

    public Object removeRear();
    //Изключва елемента накрая на текущия дек и го връща

    public Object getFirst();
    //Връща първия елемент (в началото) на текущия дек без да го изключва

    public Object getLast();
    //Връща последния елемент (накрая) на текущия дек без да го изключва
}
  
```

Да се реализират:

- Клас **StackDeque** - дек с елементи от тип **Object** с възможност за увеличаване на капацитета, съгласно спецификацията:

```

package Deque;

import java.util.EmptyStackException;
import Interface.Deque;
import Stack.FancyArrayList;
import Stack.FullStackException;

public class StackDeque implements Deque {
    //Представяне на дек чрез два стека
    protected FancyArrayList left, right;
  
```

```

//Конструктори
public StackDeque() {...} //Създава празен дек
public StackDeque(int n) {...} //Създава празен дек с начален капацитет n
public StackDeque(Object[] arg) {...} //Създава дек от елементите на масива arg, като първият
                                         //елемент на масива е в началото на дека, а последният елемент – накрая
//Реализация на методите на интерфейса Deque, така че: 1) методите removeFront, removeRear, getFirst и
//getLast активират собствено изключение EmptyDequeException, ако текущият дек е празен
//и 2) методите addFront и addRear увеличават капацитета на текущия дек, ако е необходимо

//Други методи
}

```

Реализация:

```

package Deque;

import java.util.EmptyStackException;
import Interface.Deque;
import Stack.FancyArrayStack;
import Stack.FullStackException;

public class StackDeque implements Deque {
    // Data
    protected FancyArrayStack left, right;

    // Constructor
    public StackDeque() {
        left = new FancyArrayStack();
        right = new FancyArrayStack();
    }

    public StackDeque(int n) {
        left = new FancyArrayStack(n / 2 + 1);
        right = new FancyArrayStack(n / 2 + 1);
    }

    public StackDeque(Object[] arg) {
        int n = arg.length / 2;
        left = new FancyArrayStack(2 * n);
        right = new FancyArrayStack(2 * n);
        for (int i = n - 1; i >= 0; i--)
            left.push(arg[i]);
        for (int i = n; i < arg.length; i++)
            right.push(arg[i]);
    }

    // Private method
    private void change() {
        StackDeque temp = new StackDeque(this.toArray());
        left = temp.left;
        right = temp.right;
    }

    // Deque methods implementation
    public boolean isEmpty() {
        return left.isEmpty() && right.isEmpty();
    }

    public void addFront(Object x) {
        try {
            left.push(x);
        } catch (FullStackException e) {
            change();
            left.push(x);
        }
    }

    public Object removeFront() {
        if (left.isEmpty())
            throw new EmptyStackException();
        Object result = left.pop();
        if (left.isEmpty())
            change();
        return result;
    }

    public Object removeRear() {
        if (right.isEmpty())
            throw new EmptyStackException();
        Object result = right.pop();
        if (right.isEmpty())
            change();
        return result;
    }

    public Object getFirst() {
        if (left.isEmpty())
            throw new EmptyStackException();
        return left.peek();
    }

    public Object getLast() {
        if (right.isEmpty())
            throw new EmptyStackException();
        return right.peek();
    }
}

```

```

        }

    }

    public void addRear(Object x) {
        try {
            right.push(x);
        } catch (FullStackException e) {
            change();
            right.push(x);
        }
    }

    public Object removeFront() throws EmptyDequeException {
        if (isEmpty())
            throw new EmptyDequeException("Deque is empty!");
        try {
            return left.pop();
        } catch (EmptyStackException e) {
            if (right.size() == 1)
                return right.pop();
            change();
            return left.pop();
        }
    }

    public Object removeRear() throws EmptyDequeException {
        if (isEmpty())
            throw new EmptyDequeException("Deque is empty!");
        try {
            return right.pop();
        } catch (EmptyStackException e) {
            if (left.size() == 1)
                return left.pop();
            change();
            return right.pop();
        }
    }

    public Object getFirst() throws EmptyDequeException {
        Object result = left.pop();
        left.push(result);
        return result;
    }

    public Object getLast() throws EmptyDequeException {
        Object result = right.pop();
        right.push(result);
        return result;
    }

    // Other method
    public Object[] toArray() {
        Object[] result = new Object[left.size() + right.size()];
        Object[] w1 = left.toArray();
        int n1 = w1.length;
        for (int i = 0; i < n1; i++)
            result[i] = w1[n1 - 1 - i];
        Object[] w2 = right.toArray();
        int n2 = w2.length;
        for (int i = 0; i < n2; i++)
            result[n1 + i] = w2[i];
        return result;
    }
}

```

}

Реализация на собствено изключение:

```
package Deque;

@SuppressWarnings("serial")
public class EmptyDequeException extends RuntimeException {
    public EmptyDequeException(String s) {
        super(s);
    }
}
```

2. Клас **FancyStackDeque**, съгласно спецификацията:

```
package Deque;

import Interface.FancyCollection;

public class FancyStackDeque extends StackDeque implements FancyCollection {
    //Конструктори
    public FancyStackDeque() {...} //Създава празен дек
    public FancyStackDeque(FancyCollection arg) {...} //Създава дек от елементите на колекцията arg
    public FancyStackDeque(Object[] arg) {...} //Създава дек от елементите на масива arg, като първият
                                                //елемент на масива е в началото на дека, а последният елемент - накрая
    //Наследяване на методите на интерфейса Deque
    //Реализация на методите на интерфейса FancyCollection
    //Предефиниране на методите, наследени от клас Object
}
```

Реализация:

```
package Deque;

import Interface.FancyCollection;

public class FancyStackDeque extends StackDeque implements FancyCollection {
    // Constructors
    public FancyStackDeque() {
        super();
    }

    public FancyStackDeque(Object[] arg) {
        super(arg);
    }

    public FancyStackDeque(FancyCollection arg) {
        this(arg.toArray());
    }

    // Methods - inherited from class StackDeque

    // FancyCollection methods implementation
    public int size() {
        return left.size() + right.size();
    }

    public void clear() {
        left.clear();
        right.clear();
    }

    public boolean contains(Object arg) {
        return left.contains(arg) || right.contains(arg);
    }
}
```

```

}

public Object clone() {
    return new FancyStackDeque(this);
}

// Override methods - inherited from class Object
public boolean equals(Object arg) {
    FancyStackDeque d = (FancyStackDeque) arg;
    if (this.size() != d.size())
        return false;

    Object[] thisArray = this.toArray();
    Object[] argArray = d.toArray();
    // Proverka dali masivite thisArray i argArray sа възпадат
    for (int i = 0; i < thisArray.length; i++)
        if (!thisArray[i].equals(argArray[i]))
            return false;
    return true;
}

public String toString() {
    Object[] temp = this.toArray();
    // Polu4avane na niz от elementite na masiva temp
    String result = "";
    for (int i = 0; i < temp.length; i++)
        result = result + temp[i] + '\n';
    return result;
}
}
}

```

3. Клас **PalindromChecker3** за проверка дали дума е палиндром, като се използва дек – обект на класа **StackDeque**.

Реализация:

```

package Deque;

import java.util.Scanner;
import Interface.Deque;

public class PalindromChecker3 {
    public void check(String str) {
        Deque charDeque = new StackDeque();

        for (int i = 0; i < str.length(); i++)
            charDeque.addRear(str.charAt(i));

        boolean flag = true;
        while (!charDeque.isEmpty()) {
            char charLeft = (Character) charDeque.removeFront();
            if (charDeque.isEmpty())
                break;
            char charRight = (Character) charDeque.removeRear();
            if (charLeft != charRight) {
                flag = false;
                break;
            }
        }
        System.out.println(flag ? "Yes" : "No");
    }

    class TestPalindromChecker3 {
        public static void main(String[] args) {

```

```
Scanner s = new Scanner(System.in);
System.out.println("To stop testing enter: No more");
System.out.println("Starting...");
while (true) {
    System.out.print("      Word: ");
    String word = s.nextLine();
    if (word.equals("No more")) {
        System.out.println("End of test!");
        break;
    } else
        (new PalindromChecker3()).check(word);
}
}
```

Задача: Като се използва интерфейсът **FancyCollection**, да се реализира клас **CollectionPrinter** за извеждане на елементите на колекция.

Реализация:

```
package FancyCollection;

import Stack.FancyArrayList;
import Deque.FancyStackDeque;
import Interface.FancyCollection;

public class CollectionPrinter {
    public void print(FancyCollection arg) {
        System.out.println(arg.toString());
    }
}

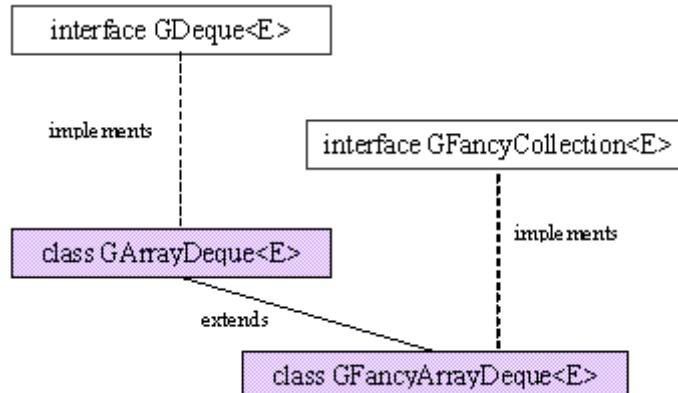
class TestCollectionPrinter {
    public static void main(String[] args) {
        CollectionPrinter w = new CollectionPrinter();

        Character[] d = { new Character('H'), new Character('e'),
                          new Character('l'), new Character('l'), new Character('o'),
                          new Character('!') };

        System.out.println("Printing stack...");
        w.print(new FancyArrayList(d));

        System.out.println("Printing deque...");
        w.print(new FancyStackDeque(d));
    }
}
```

Задача: В дадената йерархия интерфейсите **GDeque** и **GFancyCollection** са родов вариант на интерфейсите **Deque** и **FancyCollection**.



Да се реализират:

1. Клас **GArrayDeque<E>** - родов дек с ограничен капацитет, съгласно спецификацията:

```
package Deque;

import Interface.GDeque;

public class GArrayDeque<E> implements GDeque<E> {
    //Представяне на дек
    protected Object[] array;      //elements
    protected int begin;          //first element
    protected int end;             //last element
    protected int number;          //number of elements

    //Конструктори
    public GArrayDeque() {...} //Създава празен дек
    public GArrayDeque(int n) {...} //Създава празен дек с начален капацитет n
    public GArrayDeque(Object[] arg) {...} //Създава дек от елементите на масива arg, като първият
                                            //елемент на масива е в началото на дека, а последният елемент – накрая
    //Реализация на методите на интерфејса GDeque
}
```

Реализация:

```
package Deque;

import Interface.GDeque;

public class GArrayDeque<E> implements GDeque<E> {
    // Data
    protected Object[] array; // elements
    protected int begin; // first element
    protected int end; // last element
    protected int number; // number of elements

    // Constructors
    public GArrayDeque() {
        this(10);
    }

    public GArrayDeque(int n) {
        array = new Object[n];
        begin = end = -1;
        number = 0;
    }

    @SuppressWarnings("unchecked")
    public GArrayDeque(Object[] data) {
        this(data.length + 10);
        for (int i = 0; i < data.length; i++)
            addRear((E) data[i]);
    }

    // Queue methods implementation
    public boolean isEmpty() {
        return number == 0;
    }

    public void addFront(E x) throws FullDequeException {
        if (isFull())
            throw new FullDequeException("Deque is full!");
        if (isEmpty())
            begin++;
        end = (end + 1) % array.length;
        array[end] = x;
    }

    public void addRear(E x) throws FullDequeException {
        if (isFull())
            throw new FullDequeException("Deque is full!");
        if (isEmpty())
            begin = end = 0;
        else
            end = (end + 1) % array.length;
        array[end] = x;
    }

    public E removeFront() throws EmptyDequeException {
        if (isEmpty())
            throw new EmptyDequeException("Deque is empty!");
        E result = array[begin];
        begin++;
        if (begin == array.length)
            begin = 0;
        number--;
        return result;
    }

    public E removeRear() throws EmptyDequeException {
        if (isEmpty())
            throw new EmptyDequeException("Deque is empty!");
        E result = array[end];
        end--;
        if (end == -1)
            end = array.length - 1;
        number--;
        return result;
    }

    public E peekFront() {
        if (isEmpty())
            return null;
        return array[begin];
    }

    public E peekRear() {
        if (isEmpty())
            return null;
        return array[end];
    }

    public int size() {
        return number;
    }
}
```

```
    number++;
}

public void addRear(E x) throws FullDequeException {
    System.out.println("Not implemented!");
}

@SuppressWarnings("unchecked")
public E removeFront() throws EmptyDequeException {
    if (isEmpty())
        throw new EmptyDequeException("Deque is empty!");
    int i = begin;
    number--;
    if (isEmpty()) {
        begin = end = -1;
    } else
        begin = (begin + 1) % array.length;
    return (E) array[i];
}

public E removeRear() throws EmptyDequeException {
    System.out.println("Not implemented!");
    return null;
}

@SuppressWarnings("unchecked")
public E getFirst() throws EmptyDequeException {
    if (isEmpty())
        throw new EmptyDequeException("Deque is empty!");
    return (E) array[begin];
}

public E getLast() throws EmptyDequeException {
    System.out.println("Not implemented!");
    return null;
}

// Other methods
public boolean isFull() {
    return number >= array.length;
}

public int capacity() {
    return array.length;
}
```

```
Реализация на собствено изключение:  
  
package Deque;  
  
@SuppressWarnings("serial")  
public class FullDequeException extends RuntimeException {  
    public FullDequeException(String s) {  
        super(s);  
    }  
}
```

2. Клас **GFancyArrayDeque<E>**



Задача: Интерфейсът **BigInteger** представя операции на *голямо цяло десетично число без знак*:

```
package Interface;

public interface BigInteger {
    public int length();
    //Връща дължината на текущото число

    public BigInteger add(BigInteger arg);
    //Събира текущото число с числото arg

    public BigInteger mult(BigInteger arg);
    //Умножава текущото число по числото arg

    public int compareTo(BigInteger arg);
    //Сравнява текущото число с числото arg

    public boolean equals(Object obj);
    //Проверява дали текущото число и числото arg съвпадат

    public Object clone();
    //Създава копие на текущото число

    public String toString();
    //Преобразува текущото число в низ
}
```

Да се реализира клас **VectorBigInteger**, съгласно спецификацията:

```
package BigInteger;

import java.util.Vector;
import Interface.BigInteger;

public class VectorBigInteger implements BigInteger, Comparable<VectorBigInteger> {
    //Данни
    private Vector<Integer> digits; // digits = (a0,...,an)
    public static int base = 10; // base = 10^k, k = 1, 2, ...

    //Конструктори
    public VectorBigInteger() {...}
    public VectorBigInteger(BigInteger arg) {...}
    public VectorBigInteger(String arg) {...} // arg="an...a0"

    //Реализация на методите на интерфейса BigInteger
    //Реализация на метода на интерфейса Comparable
    //Предефиниране на методи, наследени от класа Object
    //Дефиниране на константи
}
```

Реализацията се основава на представянето на цифрите на голямо цяло десетично число $N = a_n \cdot base^n + a_{n-1} \cdot base^{n-1} + \dots + a_1 \cdot base + a_0$, $0 \leq a_i < base$ и $base = 10^k$, $k \geq 1$ чрез вектор $digits = (a_0, \dots, a_n)$. Използват се родовият вектор **java.util.Vector<E>** и интерфейсът **java.lang.Comparable<T>** - виж <http://java.sun.com/javase/6/docs/api/>.

```
package BigInteger;

import java.util.Vector;
import Interface.BigInteger;
```

```

public class VectorBigInteger implements BigInteger,
    Comparable<VectorBigInteger> {
    // Data
    private Vector<Integer> digits; // digits = (a0,...,an)
    public static int base = 10; // base = 10^k, k = 1, 2, ...
    // Constructors
    public VectorBigInteger() {
        this("0");
    }

    public VectorBigInteger(BigInteger arg) {
        this(arg.toString());
    }

    public VectorBigInteger(String arg) { // arg="an...a0"
        digits = new Vector<Integer>();
        int k = howLong(base) - 1;
        while (k < arg.length()) {
            int index = arg.length() - k;
            String s = arg.substring(index);
            digits.add(digits.size(), Integer.parseInt(s));
            arg = arg.substring(0, index);
        }
        digits.add(digits.size(), Integer.parseInt(arg));
    }

    // Private method
    private static int howLong(int n) {
        int d = 0;
        do {
            d++;
            n /= 10;
        } while (n != 0);
        return d;
    }

    // BigInteger methods implementation
    public int length() {
        return digits.size() * (howLong(base) - 1);
    }

    public BigInteger add(BigInteger arg) {
        VectorBigInteger Int1 = this;
        VectorBigInteger Int2 = new VectorBigInteger(arg.toString());
        Vector<Integer> left = Int1.digits;
        Vector<Integer> right = Int2.digits;

        int max = Math.max(left.size(), right.size());
        int min = Math.min(left.size(), right.size());

        VectorBigInteger w = new VectorBigInteger();
        Vector<Integer> result = w.digits = new Vector<Integer>();

        int d, c = 0;
        for (int i = 0; i < min; i++) {
            d = left.get(i) + right.get(i) + c;
            result.add(result.size(), d % base);
            c = d / base;
        }

        Vector<Integer> temp;
        if (max == left.size())
            temp = left;
        else

```

```

        temp = right;
    for (int i = min; i < temp.size(); i++) {
        d = temp.get(i) + c;
        result.add(result.size(), d % base);
        c = d / base;
    }
    if (c != 0)
        result.add(result.size(), c);

    return w;
}

public BigInteger mult(BigInteger arg) {
    VectorBigInteger Int1 = this;
    VectorBigInteger Int2 = new VectorBigInteger(arg.toString());
    Vector<Integer> left = Int1.digits;
    Vector<Integer> right = Int2.digits;

    VectorBigInteger w = new VectorBigInteger();
    Vector<Integer> result = w.digits = new Vector<Integer>();
    ;

    for (int i = 0; i < left.size() + right.size() - 1; i++)
        result.add(0, 0);

    int d, c;
    for (int i = 0; i < left.size(); i++) {
        d = left.get(i);
        for (int j = 0; j < right.size(); j++) {
            c = result.get(i + j) + d * right.get(j);
            result.set(i + j, c);
        }
    }

    c = 0;
    for (int i = 0; i < result.size(); i++) {
        d = result.get(i) + c;
        result.set(i, d % base);
        c = d / base;
    }

    if (c != 0)
        result.add(result.size(), c);

    return w;
}

public int compareTo(BigInteger arg) {
    VectorBigInteger left = this;
    VectorBigInteger right = new VectorBigInteger(arg.toString());
    return left.compareTo(right);
}

// Comparable method implementation
public int compareTo(VectorBigInteger arg) {
    String left = this.toString();
    String right = arg.toString();
    int diff = Math.abs(left.length() - right.length());
    if (left.length() < right.length())
        for (int i = 0; i <= diff; i++)
            left = "0" + left;
    else if (right.length() < left.length())
        for (int i = 0; i <= diff; i++)
            right = "0" + right;
    return left.compareTo(right);
}

```

```

// Override methods - inherited from class Object
public boolean equals(Object obj) {
    System.out.println("Not implemented!");
    return true;
}

public Object clone() {
    System.out.println("Not implemented!");
    return null;
}

public String toString() {
    String result = ""; // digits = (a0, ..., an) => result="an...a0"
    int k = howLong(base) - 1;
    for (int i = 0; i < digits.size(); i++) {
        String w = (digits.get(i)).toString();
        while (k > w.length())
            w = "0" + w;
        result = w + result;
    }
    return result;
}

// Constants
public static final VectorBigInteger ZERO = new VectorBigInteger("0");
public static final VectorBigInteger ONE = new VectorBigInteger("1");
public static final VectorBigInteger TEN = new VectorBigInteger("10");
}

```

Задача: Да се реализира родов вектор **VectorSorter<E>** с възможност за сортиране на елементите.

Реализация:

```

package Vector;

import java.util.Vector;

@SuppressWarnings("serial")
public class VectorSorter<E> extends Vector<E> {
    // Methods inherited from class Vector<E>

    // New method
    @SuppressWarnings("unchecked")
    public void sort() {
        boolean flag = true;
        while (flag) {
            flag = false;
            for (int i = 0; i < super.size() - 1; i++) {
                E e1 = super.get(i);
                E e2 = super.get(i + 1);
                if (((Comparable) e1).compareTo(e2) > 0) {
                    flag = true;
                    super.set(i, e2);
                    super.set(i + 1, e1);
                }
            }
        }
    }
}

```

Задача: Да се реализира нареден родов вектор **SortedVector<E>**.

Реализация вариант 1:

```

package Vector;
import java.util.Vector;

@SuppressWarnings("serial")
public class SortedVector<E> extends Vector<E> {
    // Methods inherited from class Vector<E>

    // New method
    @SuppressWarnings("unchecked")
    public void addInSortedVector(E x) {
        if (super.isEmpty() || ((Comparable) x).compareTo(super.get(0)) <= 0)
            super.add(0, x);
        else {
            int i;
            for (i = 0; i < super.size(); i++) {
                if (((Comparable) x).compareTo(super.get(i)) <= 0) {
                    super.add(i, x);
                    break;
                }
            }
            if (i == super.size())
                super.add(x);
        }
    }
}

```

Реализация вариант 2:

```

package Vector;
import java.util.Vector;

@SuppressWarnings("serial")
public class SortedVector2<E> extends Vector<E> {
    // Methods inherited from class Vector<E>

    // Private method
    @SuppressWarnings("unchecked")
    private int pos(E arg) {
        E midValue;
        int left = 0;
        int right = super.size();
        int mid = (left + right) / 2;
        while (left < right) {
            midValue = super.get(mid);
            if (((Comparable) midValue).compareTo(arg) < 0)
                left = mid + 1;
            else
                right = mid;
            mid = (left + right) / 2;
        }
        return left;
    }

    // Override methods
    public boolean add(E e) {
        System.out.println("My add: " + e);
        super.add(pos(e), e);
        return true;
    }

    @SuppressWarnings("unchecked")

```

```

public boolean remove(Object e) {
    System.out.println("My remove: " + e);
    int temp = pos((E) e);
    if (temp == super.size())
        return false;
    if (!e.equals(get(temp)))
        return false;
    super.remove(temp);
    return true;
}

@SuppressWarnings("unchecked")
public boolean contains(Object e) {
    System.out.println("My contains: " + e);
    int temp = pos((E) e);
    return temp != super.size() ? e.equals(get(temp)) : false;
}
}

```

Тестване:

```

package Test;

import Vector.SortedVector2;

public class SortedVector2Tester {
    public void test() {
        SortedVector2<Integer> integerVector = new SortedVector2<Integer>();

        for (int i = 6; i >= 1; i--)
            integerVector.add(i);

        System.out.println("integerVector = " + integerVector);

        integerVector.remove(3);
        System.out.println("integerVector.remove(3)");
        System.out.println("integerVector = " + integerVector);

        integerVector.remove(new Integer(3));
        System.out.println("integerVector = " + integerVector);

        System.out.println(integerVector.contains(3));
    }

    public static void main(String[] args) {
        new SortedVector2Tester().test();
    }
}

```

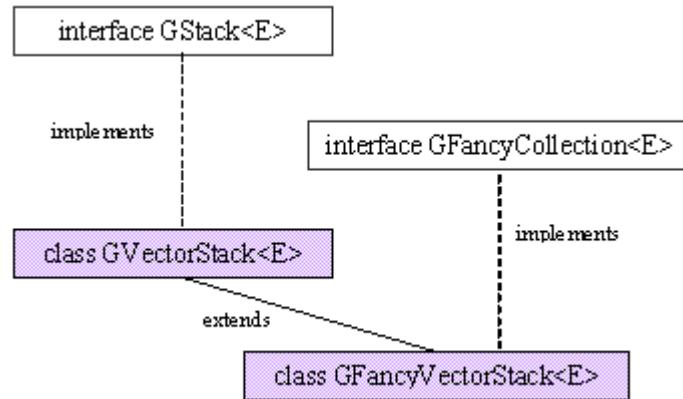
Резултати от изпълнението:

```

My add: 6
My add: 5
My add: 4
My add: 3
My add: 2
My add: 1
integerVector = [1, 2, 3, 4, 5, 6]
integerVector.remove(3)
integerVector = [1, 2, 3, 5, 6]
My remove: 3
integerVector = [1, 2, 5, 6]
My contains: 3
false

```

Задача: Дадена е йерархията:



Интерфейът **GStack** е родов вариант на интерфейса **Stack**.

Да се реализират:

- Клас **GVectorStack<E>** - родов стек с възможност за увеличаване на капацитета, съгласно спецификацията:

```

package Stack;

import java.util.EmptyStackException;
import java.util.Vector;
import Interface.GStack;

public class GVectorStack<E> implements GStack<E> {
    //Представяне на елементите
    protected Vector<E> items;

    //Конструктори
    public GVectorStack() {...}
    public GVectorStack(int n) {...}
    public GVectorStack(Object[] data) {...}

    //Реализация на методите на интерфейса GStack
}

```

Реализация:

```

package Stack;

import java.util.EmptyStackException;
import java.util.Vector;
import Interface.GStack;

public class GVectorStack<E> implements GStack<E> {
    // Data
    protected Vector<E> items;

    // Constructors
    public GVectorStack() {
        items = new Vector<E>();
    }

    public GVectorStack(int n) {
        items = new Vector<E>(n);
    }

    @SuppressWarnings("unchecked")
    public GVectorStack(Object[] data) {

```

```

    this(data.length);
    for (int i = 0; i < data.length; i++)
        push((E) data[i]);
}

// GStack methods implementation
public boolean isEmpty() {
    return items.size() == 0;
}

public void push(E x) {
    items.add(items.size(), x);
}

public E pop() throws EmptyStackException {
    if (isEmpty())
        throw new EmptyStackException();
    return items.remove(items.size() - 1);
}

public E top() throws EmptyStackException {
    if (isEmpty())
        throw new EmptyStackException();
    return items.get(items.size() - 1);
}
}

```

2. Клас GFancyVectorStack<E>, съгласно спецификацията:

```

package Stack;

import java.util.Vector;
import Interface.GFancyCollection;

public class GFancyVectorStack<E> extends GVectorStack<E> implements GFancyCollection<E>, Cloneable {
    //Конструктори
    public GFancyVectorStack() {...} //Създава празен стек
    public GFancyVectorStack(Object[] arg) {...} //Създава стек от елементите на масива arg
    public GFancyVectorStack(GFancyCollection<E> arg){...} //Създава стек от елементите на колекцията arg

    //Наследяване на методите на интерфейса GStack
    //Реализация на методите на интерфейса GFancyCollection
}

```

Реализация:

```

package Stack;

import java.util.Vector;
import Interface.GFancyCollection;

public class GFancyVectorStack<E> extends GVectorStack<E> implements
    GFancyCollection<E>, Cloneable {
    // Constructors
    public GFancyVectorStack() {
        super();
    }

    public GFancyVectorStack(Object[] arg) {
        super(arg);
    }

    public GFancyVectorStack(GFancyCollection<E> arg) {
        this(arg.toArray());
    }
}

```

```
// Methods - inherited from class GVectorStack

// GFancyCollection methods implementation
public int size() {
    return items.size();
}

public void clear() {
    items.clear();
}

public boolean contains(E arg) {
    return items.contains(arg);
}

public Object[] toArray() {
    return items.toArray();
}

@SuppressWarnings("unchecked")
public Object clone() {
    GFancyVectorStack<E> copy = null;
    try {
        copy = (GFancyVectorStack<E>) super.clone();
        copy.items = (Vector<E>) items.clone();
    } catch (java.lang.CloneNotSupportedException e) {}
    return copy;
}

@SuppressWarnings("unchecked")
public boolean equals(Object arg) {
    GFancyVectorStack<E> argStack = (GFancyVectorStack<E>) arg;
    return items.equals(argStack.items);
}

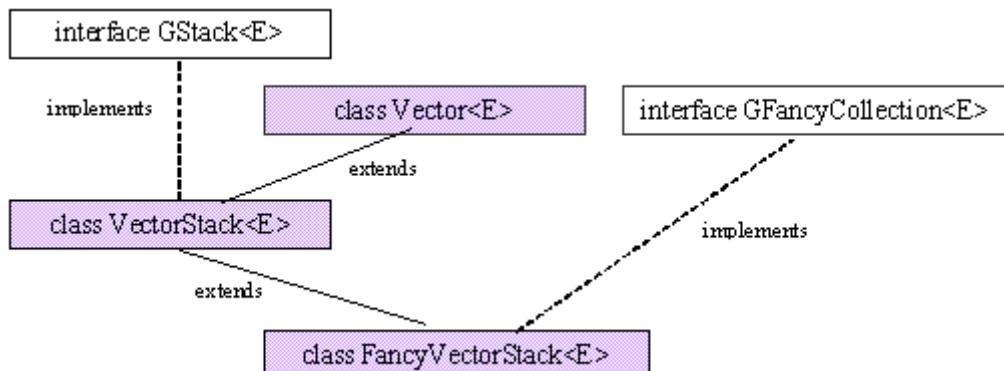
public String toString() {
    String result = "";
    for (int i = 0; i < items.size(); i++)
        result = items.get(i) + "\n" + result;
    return result;
}
}
```

Използват се родовият вектор **java.util.Vector<E>** и интерфейсът **java.lang.Cloneable** - виж <http://java.sun.com/javase/6/docs/api/>.



Задачи за упражнение

Задача: Дадена е юерархията:

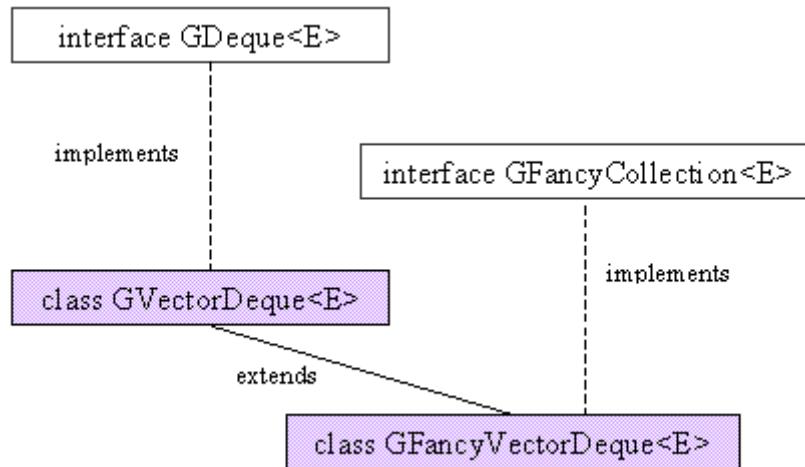


Да се реализират:

1. Клас **GVectorStack<E>** - родов стек с възможност за увеличаване на капацитета
2. Клас **GFancyVectorStack<E>**

При реализацията да се използва родовият вектор **java.util.Vector<E>**.

Задача: Дадена е йерархията:



Да се реализират:

1. Клас **GVectorDeque<E>** - родов дек с възможност за увеличаване на капацитета
2. Клас **GFancyVectorDeque<E>**
3. Клас **GPalindromChecker3** за проверка дали даден низ е палиндром, като се използва дек от символи – обект **new GVectorDeque<Character>()**



Итератор

Интерфейсът **Iterator** представя операции над обекти за обхождане на елементите на колекции, наречени *итератори*. Всеки итератор за дадена колекция „посещава“ еднократно всеки един от нейните елементи, съгласно дадено правило за обхождане.

```
public interface Iterator {
    public Object next();
    //Връща поредния непосетен елемент, ако има такъв. В противен случаи активира
//изключение NoSuchElementException

    public boolean hasNext();
    //Проверява дали има още непосетени елементи

    public void remove();
    //Изключва елемента, който последно е върнат от метода next. Ако методът
//next не е още извикан или вече е извикан методът remove, се активира
//изключение IllegalStateException. Ако методът remove не е реализиран, се
//активира изключение UnsupportedOperationException

    public void reset();
    //Подготвя текущия итератор за повторно обхождане
}
```

Следващата таблица представя различни начини за обхождане на масив, списък, дърво и таблица, както и класовете, които представляват съответните итератори.

Структура	Начин на обхождане	Клас
Масив	От елемент с даден индекс	ArrayIterator
Списък	От първия към последния елемент	LinkedListIterator DoublyLinkedListIterator
Списък	От последния към първия елемент	LinkedListReverseIterator DoublyLinkedListReverseIterator
Списък	От най-малкия елемент към най-големия или обратно (за списъци с елементи от тип Comparable)	LinkedListSortedIterator DoublyLinkedListSortedIterator
Дърво	Смесено обхождане	BSTreeInorderIterator
Дърво	Възходящо обхождане	BSTreePostorderIterator
Дърво	Низходящо обхождане	BSTreePreorderIterator
Дърво	По нива	BSTreeBreadthFirstIterator
Таблица	На ключовете	ListTableKeysIterator TreeTableKeysIterator HashTableKeysIterator
Таблица	На стойностите	ListTableValuesIterator TreeTableValuesIterator HashTableValuesIterator
Таблица	На ключовете - от най-малкия към най-големия или обратно (за таблици с ключове от тип Comparable)	ListTableKeysSortedIterator TreeTableKeysSortedIterator HashTableKeysSortedIterator

Задача: Да се реализира родов итератор **GArrayIterator<E>** за:

- Обхождане на всички елементи от първия към последния
- Циклично обхождане на даден брой елементи от елемент с даден индекс

Реализация:

```

package Iterator;

import java.util.Iterator;
import java.util.NoSuchElementException;

public class GArrayIterator<E> implements Iterator<E> {
    // Data
    private Object data[]; // Elements
    private int head; // First element
    private int count; // Number of elements
    private int current; // Current element
    private int remaining; // Number of remaining elements

    // Constructors
    public GArrayIterator(Object[] arg) {
        this(arg, 0, arg.length);
    }

    public GArrayIterator(Object[] arg, int first, int size) {
        data = arg;
        head = first;
        count = size;
        reset();
    }

    // GIterator methods implementation
    public boolean hasNext() {
        return remaining > 0;
    }

    @SuppressWarnings("unchecked")
    public E next() {
        if (hasNext()) {
            Object temp = data[current];
            current = (current + 1) % data.length;
            remaining--;
            return (E) temp;
        }
        throw new NoSuchElementException();
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }

    public void reset() {
        current = head;
        remaining = count;
    }
}

```

При реализацията се използва родовият интерфейс **java.util.Iterator<E>** - виж <http://java.sun.com/javase/6/docs/api/>.



Задача: Интерфейсът **GMultipleQueue<E>** представя операции на колекция от опашки:

```
package Interface;

import java.util.Queue;

public interface GMultipleQueue<E> {

    public boolean isEmpty();
    //Проверява дали текущата колекция е празна

    public boolean add(E object, int whichComponent);
    //Включва х в опашка с номер whichComponent ∈ [1; numberOfComponents()] на
    //текущата колекция

    public E remove(int whichComponent);
    //Изключва и връща елемент от опашка с номер
    //whichComponent ∈ [1; numberOfComponents()] на текущата колекция

    public E element(int whichComponent);
    //Връща елемент от опашка с номер whichComponent ∈ [1; numberOfComponents()]
    //на текущата колекция

    public Queue<E> getQueue(int whichComponent);
    //Връща опашка с номер whichComponent ∈ [1; numberOfComponents()] на
    //текущата колекция
    public int size();
    //Връща броя на елементите в текущата колекция

    public int numberOfComponents();
    //Връща броя на опашките в текущата колекция
}
```

Да се реализира клас **GArrayMultipleQueue<E>**, съгласно спецификацията:

```
package Queue;

import java.util.Queue;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.NoSuchElementException;
import Interface.GMultipleQueue;

public class GArrayMultipleQueue<E> implements GMultipleQueue<E> {
    //Представяне на колекция от опашки
    private Object[] array;

    //Конструктор
    public ArrayMultipleQueue(int n) {...} //Създава празна колекция от n опашки (обекти на класа
                                            //ArrayBlockingQueue<E>)
    //Реализация на методите на интерфейса MultipleQueue
}
```

Да се използват родова опашка **java.util.concurrent.ArrayBlockingQueue<E>**, реализираща интерфейса **java.util.Queue<E>** - виж <http://java.sun.com/javase/6/docs/api/>:

```
public interface Queue<E> extends Collection<E> {
    public boolean add(E e);
    // Inserts the specified element into this queue if it is possible to do so
    // immediately without violating capacity restrictions, returning true upon
    // success and throwing an IllegalStateException if no space is currently
```

```

// available

public E element();
// Retrieves, but does not remove, the head of this queue
// Throws: NoSuchElementException if this queue is empty

public E remove();
// Retrieves and removes the head of this queue
// Throws: NoSuchElementException if this queue is empty

public boolean offer(E e);
// Inserts the specified element into this queue if it is possible to do so
// immediately without violating capacity restrictions
// Returns: true if the element was added to this queue, else false

public E peek();
// Retrieves, but does not remove, the head of this queue, or returns null
// if this queue is empty

public E poll();
// Retrieves and removes the head of this queue, or returns null
// if this queue is empty
}

```

Реализация:

```

package Queue;

import java.util.Queue;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.NoSuchElementException;
import Interface.GMultipleQueue;

public class GArrayMultipleQueue<E> implements GMultipleQueue<E> {
    // Data
    private Object[] array;

    // Constructor
    public GArrayMultipleQueue(int n) {
        array = new Object[n];
        for (int i = 0; i < n; i++)
            array[i] = new ArrayBlockingQueue<E>(100);
    }

    // Private method
    private void check(int i) throws InvalidNumberOfComponent {
        if (i < 1 || i > numberComponents())
            throw new InvalidNumberOfComponent("Invalid number of components!" + i);
    }

    // Public methods
    @SuppressWarnings("unchecked")
    public boolean isEmpty() {
        for (int i = 0; i < array.length; i++)
            if (!((Queue<E>) array[i]).isEmpty())
                return false;
        return true;
    }

    @SuppressWarnings("unchecked")
    public boolean add(E object, int whichComponent)
        throws InvalidNumberOfComponent, IllegalStateException {
        check(whichComponent);
        return ((Queue<E>) array[whichComponent - 1]).add(object);
    }
}

```

```

}

@SuppressWarnings("unchecked")
public E remove(int whichComponent) throws InvalidNumberOfComponent,
    NoSuchElementException {
    check(whichComponent);
    return ((Queue<E>) array[whichComponent - 1]).remove();
}

@SuppressWarnings("unchecked")
public E element(int whichComponent) throws InvalidNumberOfComponent,
    NoSuchElementException {
    check(whichComponent);
    return ((Queue<E>) array[whichComponent - 1]).element();
}

@SuppressWarnings("unchecked")
public Queue<E> getQueue(int whichComponent)
    throws InvalidNumberOfComponent {
    check(whichComponent);
    return (Queue<E>) array[whichComponent - 1];
}

@SuppressWarnings("unchecked")
public int size() {
    int result = 0;
    for (int i = 0; i < array.length; i++)
        result = result + ((Queue<E>) array[i]).size();
    return result;
}

public int numberOfComponents() {
    return array.length;
}
}
}

```

Реализация на собствено изключение:

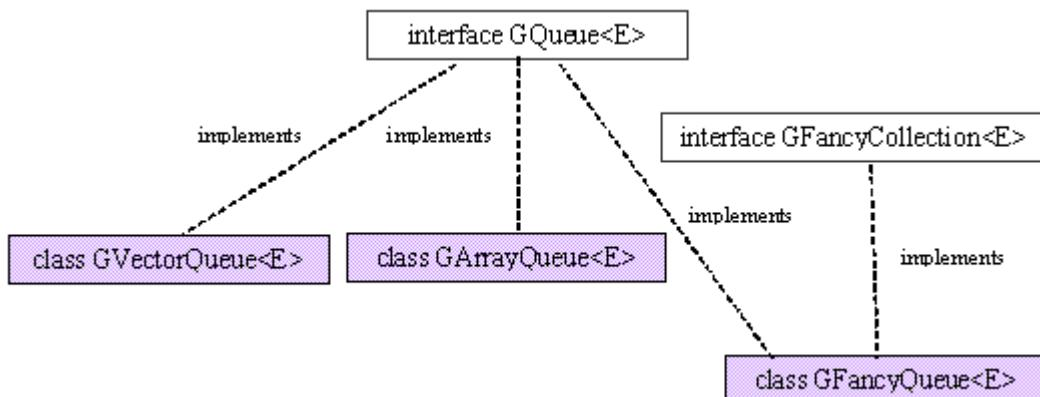
```

package Queue;

@SuppressWarnings("serial")
public class InvalidNumberOfComponent extends RuntimeException {
    InvalidNumberOfComponent(String arg) {
        super(arg);
    }
}

```

Задача: Дадена е ѿрархията:



Интерфейсът **GQueue<E>** представя операции на опашка:

```

package Interface;

public interface GQueue<E> {
    public boolean isEmpty();

    public boolean add(E e);

    public E element();

    public E remove();

    public boolean offer(E e);

    public E peek();

    public E poll();

    public Object[] toArray();

    public Object clone();
}

```

Да се реализират:

1. Клас **GArrayQueue<E>** - родова опашка с ограничен капацитет, съгласно спецификацията:

```

package Queue;

import java.util.NoSuchElementException;
import java.util.Iterator;
import Iterator.GArrayIterator;
import Interface.GQueue;

public class GArrayQueue<E> implements GQueue<E>, Cloneable {
    //Представяне на опашка чрез масив
    protected Object[] array;      //Кръгов буфер за елементите – виж List of data structures
    protected int begin;          //Индекс на първия елемент
    protected int end;            //Индекс на последния елемент
    protected int number;         //Брой на елементите

    //Конструктори
    public GArrayQueue() {...} //Създава празна опашка
    public GArrayQueue(int n) {...} //Създава празна опашка с капацитет n
    public GArrayQueue(Object[] arg) {...} //Създава опашка от елементите на масива arg, като първият
        //елемент на масива е в началото на опашката, а последният елемент – накрая на опашката
    //Реализация на методите на интерфейса GQueue
    //Други методи
    public boolean isFull() {...}
    public int capacity() {...}
    public Iterator<E> iterator() {...} //Итератор за обхождане на елементите на опашката от първия към
        //последния
    //Предефиниране на метод, наследен от клас Object
    public Object clone() {...}
}

```

Реализация:

```

package Queue;

import java.util.Iterator;
import java.util.NoSuchElementException;
import Interface.GQueue;
import Iterator.GArrayIterator;

```

```

public class GArrayQueue<E> implements GQueue<E>, Cloneable {
    // Data
    protected Object[] array; // Elements
    protected int begin; // First element
    protected int end; // Last element
    protected int number; // Number of elements

    // Constructors
    public GArrayQueue() {
        this(10);
    }

    public GArrayQueue(int n) {
        array = new Object[n];
        begin = 0;
        end = -1;
        number = 0;
    }

    @SuppressWarnings("unchecked")
    public GArrayQueue(Object[] data) {
        this(data.length + 10);
        for (int i = 0; i < data.length; i++)
            add((E) data[i]);
    }

    // GQueue methods implementation
    public boolean isEmpty() {
        return number == 0;
    }

    public boolean add(E x) throws IllegalStateException {
        if (isFull())
            throw new IllegalStateException();
        end = (end + 1) % array.length;
        array[end] = x;
        number++;
        return true;
    }

    @SuppressWarnings("unchecked")
    public E element() throws NoSuchElementException {
        if (isEmpty())
            throw new NoSuchElementException();
        return (E) array[begin];
    }

    public E remove() throws NoSuchElementException {
        E result = this.element();
        number--;
        array[begin] = null;
        begin = (begin + 1) % array.length;
        return result;
    }

    public boolean offer(E e) {
        try {
            this.add(e);
            return true;
        } catch (IllegalStateException ex) {
            return false;
        }
    }

    public E peek() {

```

```

try {
    return this.element();
} catch (NoSuchElementException e) {
    return null;
}

public E poll() {
    try {
        return this.remove();
    } catch (NoSuchElementException e) {
        return null;
    }
}

public Object[] toArray() {
    Object[] result = new Object[number];
    if (!isEmpty()) {
        int i = begin;
        int pos = 0;
        while (true) {
            result[pos++] = array[i];
            if (i == end)
                break;
            i = (i + 1) % array.length;
        }
    }
    return result;
}

// Other methods
public boolean isFull() {
    return number >= array.length;
}

public int capacity() {
    return array.length;
}

// Override method
@SuppressWarnings("unchecked")
public Object clone() {
    GArrayQueue<E> copy = null;
    try {
        copy = (GArrayQueue<E>) super.clone();
        copy.array = this.array.clone();
        copy.begin = this.begin;
        copy.end = this.end;
        copy.number = this.number;
    } catch (java.lang.CloneNotSupportedException e) {
    }
    return copy;
}

public Iterator<E> iterator() {
    return new GArrayIterator<E>(array, begin, number);
}
}

```

2. Клас **GVectorQueue<E>** - родова опашка с възможност за увеличаване на капацитета, съгласно спецификацията:

```

package Queue;

import java.util.NoSuchElementException;
import java.util.Iterator;

```

```

import Interface.GQueue;

public class GVectorQueue<E> implements GQueue<E>, Cloneable {
    //Представяне на опашка чрез вектор
    private Vector<E> elements;

    //Конструктори
    public GVectorQueue() {...} //Създава празна опашка
    public GVectorQueue(int n) {...} //Създава празна опашка с капацитет n
    public GVectorQueue(Object[] arg) {...} //Създава опашка от елементите на масива arg, като първият
        //елемент на масива е в началото на опашката, а последният елемент – накрая на опашката
    //Реализация на методите на интерфейса GQueue
    //Други методи
    public int capacity() {...}
    public Iterator<E> iterator() {...} //Итератор за обхождане на елементите на опашката от първия към
        //последния
    //Предефиниране на метод, наследен от клас Object
    public Object clone() {...}
}

```

3. Клас **GFancyQueue<E>**, съгласно спецификацията:

```

package Queue;

import java.util.NoSuchElementException;
import Interface.GFancyCollection;
import Interface.GQueue;

public class GFancyQueue<E> implements GQueue<E>, GFancyCollection<E>, Cloneable {
    //Представяне
    private GQueue<E> elements;

    //Конструктори
    public GFancyQueue(GQueue<E> arg) {...} //Създава колекция-опашка, съвпадаща с опашката arg

    //Реализация на методите на интерфейса GQueue
    //Реализация на методите на интерфейса GFancyCollection
    //Предефиниране на методите, наследени от клас Object
}

```



Задачи за упражнение

Задача: Като се използва **java.util.concurrent.ArrayBlockingQueue<E>**, да се реализират:

1. Клас **IntegersPrinter2** за генериране на случаен брой от случайни цели числа и извеждане на числата в реда на получаването им
2. Клас **TextPrinter2** за въвеждане на текст и извеждане на първите n (случайно число) реда от него

Задача: Като се използва **GArrayMultipleQueue<E>**, да се реализират:

1. Клас **TextPrinter3** за въвеждане на последователност от думи и тяхното извеждане в азбучен ред на първите им букви. Думите, започващи с една и съща буква, се извеждат в реда на въвеждането им
2. Клас **IntegersPrinter3** за въвеждане на последователност от цели положителни числа и тяхното извеждане в намаляващ ред на дълчините им. Числата с една и съща дължина се извеждат в реда на въвеждането им



Задача: Да се реализира нареден родов списък **SortedLinkedList<E>**.

Реализация:

```
package List;

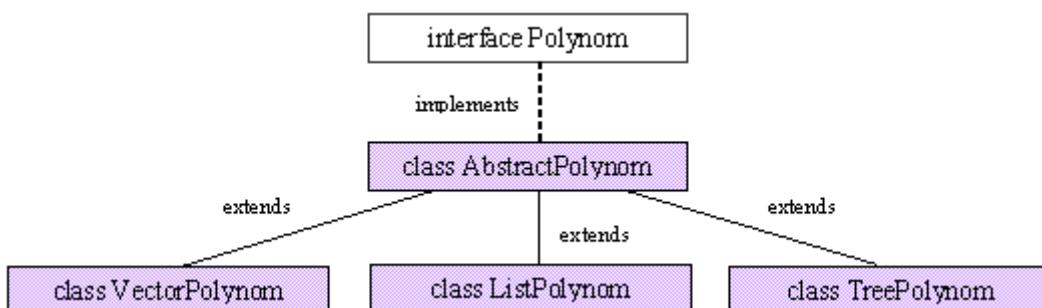
import java.util.Comparator;
import java.util.LinkedList;

@SuppressWarnings("serial")
public class SortedLinkedList<E> extends LinkedList<E> {
    // Methods inherited from class LinkedList<E>

    // New method
    public void addInSortedList(Comparator<E> c, E x) {
        if (super.isEmpty() || c.compare(x, super.get(0)) <= 0)
            super.add(0, x);
        else {
            int i;
            for (i = 0; i < super.size(); i++) {
                if (c.compare(x, super.get(i)) <= 0) {
                    super.add(i, x);
                    break;
                }
            }
            if (i == super.size())
                super.add(x);
        }
    }
}
```

Използват се родовият свързан списък **java.util.LinkedList<E>** и интерфейсът **java.util.Comparator<T>** - виж <http://java.sun.com/javase/6/docs/api/>.

Задача: Дадена е йерархията от интерфейси и класове:



Интерфейсът **Polynom** представя операции на *полином на една променлива с коефициенти реални числа*:

```
package Interface;

import java.util.Iterator;
import Polynom.Pair;

public interface Polynom {
    public int degree();
    //Връща най-високата степен на едночлен в текущия полином.
```

```

public double coeff(int degree);
//Връща коефициента пред степен degree в текущия полином

public double setCoeff(int degree, double value);
//Променя коефициента пред степен degree в текущия полином на value и връща
//старата стойност

public Polynom add(Polynom arg);
//Определя сумата на текущия полином и arg

public Polynom mult(Polynom arg);
//Определя произведението на текущия полином и arg

public Polynom derive();
//Определя първата производна на текущия полином

public double eval(double value);
//Пресмята стойността на текущия полином за стойността value

public double[] toArray();
//Връща масив с коефициентите на текущия полином и степени - индексите
//на масива

public boolean equals(Object obj);
//Проверява дали текущия полином и полинома obj съвпадат

public String toString();
//Връща текущия полином, представен в символен вид

public Iterator<Pair> iterator();
//Итератор за обхождане на членовете на текущия полином с ненулеви
//коефициенти в намаляващ ред на степените
}

```

Кластьт **Pair** представя двойка (кофициент, степен):

```

package Polynom;

public class Pair implements Comparable<Pair> {
    // Data
    private double d;
    private int i;

    // Constructor
    public Pair(double d, int i) {
        this.d = d;
        this.i = i;
    }

    // Access methods
    public double getDouble() {
        return d;
    }

    public double setDouble(double value) {
        double w = d;
        d = value;
        return w;
    }

    public int getInteger() {
        return i;
    }

    public int setInteger(int value) {

```

```

    int w = i;
    i = value;
    return w;
}

// Comparable method implementation
public int compareTo(Pair obj) {
    return obj.i - this.i;
}

// Override method - inherited from class Object
public boolean equals(Object obj) {
    Pair w = (Pair) obj;
    return i == w.i && d == w.d;
}

public String toString() {
    return this.d + " * x ^ " + this.i;
}
}
}

```

Класът **NaturalComparator<E>** представя обект за сравняване на два обекта, използвайки метода `compareTo`:

```

package Comparator;

import java.util.Comparator;

public class NaturalComparator<E extends Comparable<E>> implements
    Comparator<E> {

    public int compare(E a, E b) {
        return a.compareTo(b);
    }
}

```

Да се реализират:

1. Клас **AbstractPolynom**, съгласно спецификацията:

```

package Polynom;

import java.util.Iterator;
import Interface.Polynom;

public abstract class AbstractPolynom implements Polynom {
    //Степен на полином
    protected int degree;

    //Абстрактни методи
    public abstract double coeff(int degree);
    public abstract double setCoeff(int degree,double value);
    protected abstract Polynom create(); //Създава полинома P(x)=0
    public abstract Iterator<Pair> iterator();

    //Реализация на методи на интерфейса Polynom
    public int degree() {...}
    public Polynom add(Polynom arg) {...}
    public Polynom mult(Polynom arg) {...}
    public double eval(double value) {...}
    public Polynom derive() {...}
    public double[] toArray() {...}

    //Предефиниране на наследени методи от клас Object
    public boolean equals(Object arg) {...}
}

```

```
    public String toString() {...}
}
```

Реализация:

```
package Polynom;

import java.util.Iterator;
import Interface.Polynom;

public abstract class AbstractPolynom implements Polynom {
    // Data
    protected int degree;

    // Abstract methods
    public abstract double coeff(int degree);

    public abstract double setCoeff(int degree, double value);

    protected abstract Polynom create();

    public abstract Iterator<Pair> iterator();

    // Polynom methods
    public int degree() {
        return this.degree;
    }

    public Polynom add(Polynom arg) {
        Polynom result = create();
        int degree = Math.max(this.degree(), arg.degree());
        for (int i = 0; i <= degree; i++) {
            result.setCoeff(i, this.coeff(i) + arg.coeff(i));
        }
        return result;
    }

    public Polynom mult(Polynom arg) {
        System.out.println("Not implemented!");
        return null;
    }

    public double eval(double value) {
        Iterator<Pair> it = this.iterator();
        double result = 0.0;
        while (it.hasNext()) {
            Pair p = it.next();
            result += p.getDouble() * Math.pow(value, p.getInteger());
        }
        return result;
    }

    public Polynom derive() {
        Iterator<Pair> it = this.iterator();
        Polynom result = create();
        while (it.hasNext()) {
            Pair p = it.next();
            if (p.getInteger() != 0)
                result.setCoeff(p.getInteger() - 1, p.getDouble()
                    * p.getInteger());
        }
        return result;
    }

    public double[] toArray() {
        Iterator<Pair> it = this.iterator();
```

```

double[] result = new double[this.degree() + 1];
while (it.hasNext()) {
    Pair p = it.next();
    result[p.getInteger()] = p.getDouble();
}
return result;
}

// Override methods - inherited from class Object
public boolean equals(Object arg) {
    Polynom P = this;
    Polynom Q = (Polynom) arg;
    if (P.degree() != Q.degree())
        return false;
    double[] PArray = P.toArray();
    double[] QArray = Q.toArray();
    // Proverka dali masivite PArray i QArray savpadat
    for (int i = 0; i < PArray.length; i++)
        if (PArray[i] != QArray[i])
            return false;
    return true;
}

public String toString() {
    String result = "Polynom degree: " + this.degree + '\n';
    Iterator<Pair> it = this.iterator();
    while (it.hasNext()) {
        Pair p = it.next();
        int degree = p.getInteger();
        double coeff = p.getDouble();
        result = result + "Coefficient: " + coeff + " Degree: " + degree
            + '\n';
    }
    return result;
}
}

```

2. Клас **VectorPolynom**, съгласно спецификацията:

```

package Polynom;

import java.util.Iterator;
import Vector.SortedVector;
import Interface.Polynom;

public class VectorPolynom extends AbstractPolynom implements Polynom {
    //Представяне на полином
    private SortedVector<Pair> coeffs;      //Ненулеви коефициенти

    //Конструктори
    public VectorPolynom(double coeff,int degree) {...} //P(x)=coeff . x^degree
    public VectorPolynom(double[] coeffs){...} //P(x)=coeffs[0]+...+coeffs[coeffs.length-1]x**(coeffs.length-1)
    public VectorPolynom(Polynom p) {...} //P(x)=p

    //Собствен метод
    private double remove(int degree) {...} //Премахва едночлен от степен degree от текущия полином, като
                                              //връща коефициента
    //Реализация на абстрактните методи на класа AbstractPolynom
    //Наследяване на методите на интерфейса Polynom, реализирани в класа AbstractPolynom
}

```

Реализацията на класа **VectorPolynom** се основава на представяне на коефициентите на полином $P(x) = a_n x^n + \dots + a_0$ чрез вектор **coeffs** = {(a_i, i) | a_i ≠ 0}, нареден в намаляващ ред на степените, като се използва класът **SortedVector<E>**.

3. Клас **ListPolynom**, съгласно спецификацията:

```

package Polynom;

import java.util.Iterator;
import Comparator.NaturalComparator;
import Interface.Polynom;
import List.SortedLinkedList;

public class ListPolynom extends AbstractPolynom implements Polynom {
    //Представяне на полином
    private SortedLinkedList<Pair> coeffs; //Ненулеви коефициенти

    //Конструктори
    public ListPolynom(double coeff,int degree) {...} //P(x)= coeff . x^degree
    public ListPolynom(double[] coeffs){...} //P(x)=coeffs[0]+...+ coeffs[coeffs.length-1]x**(coeffs.length-1)
    public ListPolynom(Polynom p) {...} //P(x)=p

    //Собствен метод
    private double remove(int degree) {...} //Премахва едночлен от степен degree от текущия полином, като
                                              //връща коефициента
    //Реализация на абстрактните методи на класа AbstractPolynom
    //Наследяване на методите на интерфейса Polynom, реализирани в класа AbstractPolynom
}

```

Реализацията на класа **ListPolynom** се основава на представяне на коефициентите на полином $P(x) = a_n x^n + \dots + a_0$ чрез списък **coeffs** = $\{(a_i, i) \mid a_i \neq 0\}$, нареден в намаляващ ред на степените, като се използва класът **SortedLinkedList<E>**.

```

package Polynom;

import java.util.Iterator;
import Comparator.NaturalComparator;
import Interface.Polynom;
import List.SortedLinkedList;

public class ListPolynom extends AbstractPolynom implements Polynom {
    // Data
    private SortedLinkedList<Pair> coeffs;

    // Constructors
    public ListPolynom(double coeff, int degree) {
        this.coeffs = new SortedLinkedList<Pair>();
        this.degree = 0; // 0x**0
        setCoeff(degree, coeff);
    }

    public ListPolynom(double[] coeffs) {
        this(0.0, 0);
        for (int i = coeffs.length - 1; i >= 0; i--)
            setCoeff(i, coeffs[i]);
    }

    public ListPolynom(Polynom p) {
        ListPolynom sum = (ListPolynom) ((new ListPolynom(0.0, 0)).add(p));
        this.coeffs = sum.coeffs;
        this.degree = sum.degree;
    }

    // Private method
    private double remove(int degree) {
        double coeff = coeff(degree);
        if (coeff != 0.0) {

```

```

        coeffs.remove(new Pair(coeff, degree));
        if (this.degree == degree && !coeffs.isEmpty())
            this.degree = coeffs.get(0).getInteger();
    }
    return coeff;
}

// Abstract methods implementation
public double coeff(int degree) {
    Iterator<Pair> it = coeffs.iterator();
    while (it.hasNext()) {
        Pair p = it.next();
        if (p.getInteger() == degree)
            return p.getDouble();
    }
    return 0.0;
}

public double setCoeff(int degree, double value) {
    double result = remove(degree);
    if (value != 0) {
        coeffs.addInSortedList(new NaturalComparator<Pair>(), new Pair(
            value, degree));
        if (super.degree < degree)
            super.degree = degree;
    }
    return result;
}

protected Polynom create() {
    return new ListPolynom(0.0, 0);
}

public Iterator<Pair> iterator() {
    Iterator<Pair> it = coeffs.iterator();
    return it;
}

// Polynom methods - inherited from AbstractPolynom
}

```

4. Клас TreePolynom, съгласно спецификацията:

```

package Polynom;

import java.util.Iterator;
import java.util.NoSuchElementException;
import Interface.Polynom;
import Tree.BinarySearchTree;

public class TreePolynom extends AbstractPolynom implements Polynom {
    //Представяне на полином
    private BinarySearchTree coeffs; //Ненулеви коефициенти

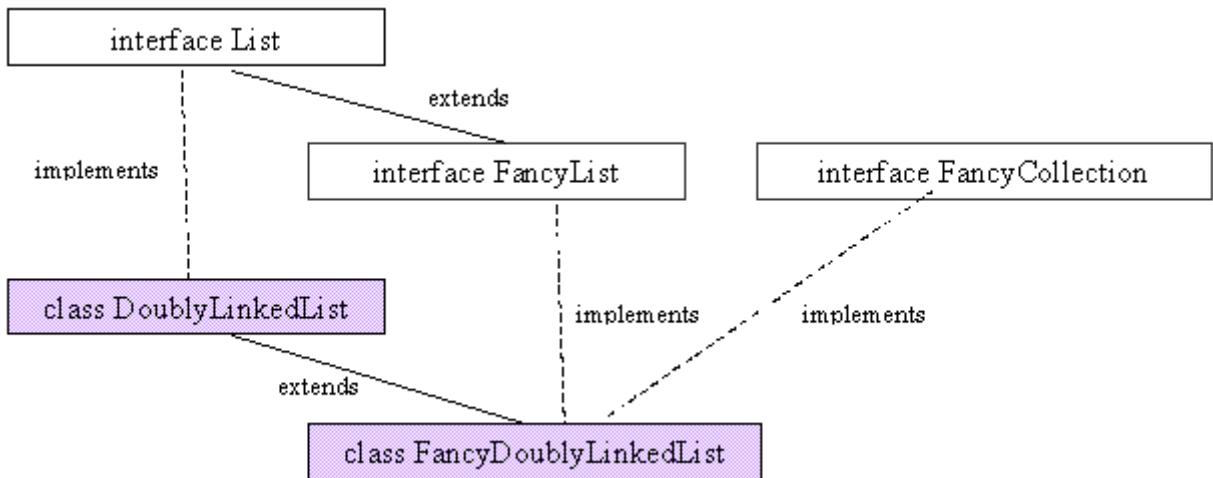
    //Конструктори
    public TreePolynom(double coeff,int degree) {...} //P(x)= coeff . x^degree
    public TreePolynom(double[] coeffs){...} //P(x)=coeffs[0]+...+ coeffs[coeffs.length-1]x**(coeffs.length-1)
    public TreePolynom(Polynom p) {...} //P(x)=p

    //Собствен метод
    private double remove(int degree) {...} //Премахва едночлен от степен degree от текущия полином, като
                                              //връща коефициента
    //Реализация на абстрактните методи на класа AbstractPolynom
    //Наследяване на методите на интерфейса Polynom, реализирани в класа AbstractPolynom
}

```

Реализацията на класа **TreePolynom** се основава на представяне на коефициентите на полином $P(x) = a_n x^n + \dots + a_0$ чрез двоично дърво $\text{coeffs} = \{(a_i, i) \mid a_i \neq 0\}$, наредено в намаляващ ред на степените, като се използва класът **BinarySearchTree** от Тема 14.

Задача: Дадена е йерархията от интерфейси и класове:



Интерфейсът **List** представя операции на *списък с елементи от тип Object*:

```

package Interface;

import java.util.Iterator;

public interface List {
    public boolean add(Object element);

    public void add(int index, Object element);

    public void addFirst(Object element);

    public void addLast(Object element);

    public boolean contains(Object arg);

    public Object get(int index);

    public Object getFirst();

    public Object getLast();

    public int indexOf(Object element);

    public int lastIndexOf(Object element);

    public boolean isEmpty();

    @SuppressWarnings("unchecked")
    public Iterator iterator();

    public Object remove(int index);

    public Object remove(Object element);

    public Object removeFirst();

    public Object removeLast();
}
  
```

```

public Object set(int index, Object element);

public int size();

public String toString();
}

```

Интерфейсът **FancyList** представя операции над списъци:

```

package Interface;

public interface FancyList extends List {
    public boolean containsAll(FancyList c);
    // Returns true if this list contains all of the elements
    // of the specified collection

    public boolean addAll(FancyList c);
    // Appends all of the elements in the specified collection to the end of
    // this list,
    // in the order that they are returned by the specified collection's
    // iterator
    // Returns true if this list changed as a result of the call
    // Throws:
    // UnsupportedOperationException - if the addAll operation is not supported
    // by this list

    public boolean addAll(int index, FancyList c);
    // Inserts all of the elements in the specified collection into this list at
    // the
    // specified position. Shifts the element currently at that position (if
    // any) and
    // any subsequent elements to the right (increases their indices). The new
    // elements
    // will appear in this list in the order that they are returned by the
    // specified
    // collection's iterator
    // Returns true if this list changed as a result of the call
    // Throws:
    // UnsupportedOperationException - if the addAll operation is not supported
    // by this list
    // IndexOutOfBoundsException - if the index is out of range (index < 0 ||
    // index > size())

    public boolean removeAll(FancyList c);
    // Removes from this list all of its elements that are contained in the
    // specified
    // collection
    // Returns true if this list changed as a result of the call
    // Throws:
    // UnsupportedOperationException - if the removeAll operation is not
    // supported by this list

    public boolean retainAll(FancyList c);
    // Retains only the elements in this list that are contained in the
    // specified collection.
    // In other words, removes from this list all the elements that are not
    // contained in
    // the specified collection
    // Returns true if this list changed as a result of the call
    // Throws:
    // UnsupportedOperationException - if the retainAll operation is not
    // supported by this list

    public FancyList subList(int fromIndex, int toIndex);
    // Returns a view of the portion of this list between the specified
    // fromIndex, inclusive,
}

```

```
// and toIndex, exclusive. If fromIndex and toIndex are equal, the returned
// list is empty
// Throws:
// IndexOutOfBoundsException - if (fromIndex < 0 || toIndex > size() ||
// fromIndex > toIndex)
}
```

Да се реализират:

1. Клас **DoublyLinkedList** - линеен двусвързан списък с елементи от тип **Object**, съгласно спецификацията:

```
package List;

import java.util.Iterator;
import Interface.List;
import Iterator.DoublyLinkedListIterator;
import Iterator.DoublyLinkedListReverseIterator;

public class DoublyLinkedList implements List {
    //Представяне
    private DNode head,tail;
    protected int num;

    //Методи за достъп
    public DNode getHead() { return head; }
    public void setHead(DNode p) { head = p; }
    public DNode getTail() { return tail; }
    public void setTail(DNode p) { tail = p; }
    public void setNum(int n) { num = n; }
    public int getNum() { return num; }

    //Конструктор, който създава празен списък
    public DoublyLinkedList(){
        setHead(new DNode());
        setTail(new DNode());
        getTail().setLeft(head);
        getHead().setRight(tail);
        num = 0;
    }

    //Метод, който връща указател към възела с индекс index на текущия списък
    protected DNode search(int index) throws IndexOutOfBoundsException {
        if(index < 0 || index >= size()) throw new IndexOutOfBoundsException();
        DNode p = getHead().getRight();
        for(int i = 0; i < index; i++) p = p.getRight();
        return p;
    }

    //Реализация на методите на интерфейса List
    //...
    public Iterator iterator() { return new DoublyLinkedListIterator(this); }

    //Друг метод
    public Iterator reverseIterator() { return new DoublyLinkedListReverseIterator(this); }
}
```

Реализация:

```
package List;

import java.util.Iterator;
import Interface.List;
import Iterator.DoublyLinkedListIterator;
import Iterator.DoublyLinkedListReverseIterator;
```

```

public class DoublyLinkedList implements List {
    // Data
    private DNode head, tail;
    protected int num;

    // Access methods
    public DNode getHead() {
        return head;
    }

    public DNode getTail() {
        return tail;
    }

    // Constructor
    public DoublyLinkedList() {
        head = new DNode();
        tail = new DNode();
        tail.setLeft(head);
        head.setRight(tail);
        num = 0;
    }

    // Private methods
    protected DNode search(int index) throws IndexOutOfBoundsException {
        if (index < 0 || index >= size())
            throw new IndexOutOfBoundsException();
        DNode p = getHead().getRight();
        for (int i = 0; i < index; i++)
            p = p.getRight();
        return p;
    }

    // List methods
    public boolean add(Object element) {
        addLast(element);
        return true;
    }

    public void add(int index, Object obj) throws IndexOutOfBoundsException {
        if (index == 0)
            addFirst(obj);
        else if (index == size())
            addLast(obj);
        else {
            DNode p = search(index);
            DNode w = new DNode(obj);
            w.setRight(p);
            w.setLeft(p.getLeft());
            p.getLeft().setRight(w);
            p.setLeft(w);
            num++;
        }
    }

    public void addFirst(Object o) {
        DNode p = new DNode(o);
        p.setLeft(getHead());
        p.setRight(getHead().getRight());
        getHead().setRight(p);
        p.getRight().setLeft(p);
        num++;
    }

    public void addLast(Object o) {

```

```

DNode p = new DNode(o);
p.setRight(getTail());
p.setLeft(getTail().getLeft());
getTail().setLeft(p);
p.getLeft().setRight(p);
num++;
}

@SuppressWarnings("unchecked")
public boolean contains(Object arg) {
    Iterator it = iterator();
    while (it.hasNext())
        if (arg.equals(it.next()))
            return true;
    return false;
}

public Object get(int index) {
    return search(index).getData();
}

public Object getFirst() {
    return get(0);
}

public Object getLast() {
    return get(size() - 1);
}

public int indexOf(Object element) {
    throw new UnsupportedOperationException();
}

public int lastIndexOf(Object element) {
    throw new UnsupportedOperationException();
}

public boolean isEmpty() {
    return num == 0;
}

@SuppressWarnings("unchecked")
public Iterator iterator() {
    return new DoublyLinkedListIterator(this);
}

public Object remove(int index) throws IndexOutOfBoundsException {
    if (index == 0)
        return removeFirst();
    if (index == size() - 1)
        return removeLast();
    DNode p = search(index);
    Object d = p.getData();
    p.getRight().setLeft(p.getLeft());
    p.getLeft().setRight(p.getRight());
    num--;
    return d;
}

public Object remove(Object element) {
    if (isEmpty())
        return null;
    DNode p = getHead().getRight();
    for (int i = 0; i < size(); i++)
        if (element.equals(p.getData()))
            return remove(i);
}

```

```

    else
        p = p.getRight();
    return null;
}

public Object removeFirst() {
    Object o = null;
    if (!isEmpty()) {
        DNode p = getHead().getRight();
        getHead().setRight(p.getRight());
        p.getRight().setLeft(getHead());
        o = p.getData();
        num--;
    }
    return o;
}

public Object removeLast() {
    Object o = null;
    if (!isEmpty()) {
        DNode p = getTail().getLeft();
        getTail().setLeft(p.getLeft());
        p.getLeft().setRight(getTail());
        o = p.getData();
        num--;
    }
    return o;
}

@SuppressWarnings("unchecked")
public Iterator reverseIterator() {
    return new DoublyLinkedListReverseIterator(this);
}

public Object set(int index, Object obj) {
    DNode p = search(index);
    Object result = p.getData();
    p.setData(obj);
    return result;
}

public int size() {
    return num;
}

@SuppressWarnings("unchecked")
public String toString() {
    String result = "[";
    Iterator it = this.iterator();
    while (it.hasNext())
        result = result + it.next() + " ";
    return result + "]";
}
}

```

Класът **DNode** представя възел на линеен двусвързан списък:

```

package List;

public class DNode {
    // Data
    private Object data;
    private DNode left, right;

    // Constructors
    public DNode() {

```

```

        data = left = right = null;
    }

public DNode (Object d) {
    data = d;
    left = right = null;
}

// Methods
public DNode getLeft() {
    return left;
}

public void setLeft(DNode l) {
    left = l;
}

public DNode getRight() {
    return right;
}

public void setRight(DNode r) {
    right = r;
}

public Object getData() {
    return data;
}

public void setData(Object d) {
    data = d;
}
}

```

Класът **DoublyLinkedListIterator** представя итератор за обхождане на списък от първия към последния му елемент:

```

package Iterator;

import java.util.Iterator;
import java.util.NoSuchElementException;
import List.DNode;
import List.DoublyLinkedList;

@SuppressWarnings("unchecked")
public class DoublyLinkedListIterator implements Iterator {
    // Data
    DoublyLinkedList list;
    DNode current;

    // Constructor
    public DoublyLinkedListIterator(DoublyLinkedList list) {
        this.list = list;
        current = list.getHead();
    }

    // Methods
    public boolean hasNext() {
        return current.getRight() != list.getTail();
    }

    public Object next() {
        if (hasNext()) {
            current = current.getRight();
            return current.getData();
        }
    }
}

```

```

        }
        throw new NoSuchElementException();
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }

    public void reset() {
        current = list.getHead();
    }
}

```

Класът **DoublyLinkedListReverseIterator** представя итератор за обхождане на списък от последния към първия му елемент:

```

package Iterator;

import java.util.Iterator;
import java.util.NoSuchElementException;
import List.DNode;
import List.DoublyLinkedList;

@SuppressWarnings("unchecked")
public class DoublyLinkedListReverseIterator implements Iterator {
    // Data
    DoublyLinkedList list;
    DNode current;

    // Constructor
    public DoublyLinkedListReverseIterator(DoublyLinkedList list) {
        this.list = list;
        current = list.getTail();
    }

    // Methods
    public boolean hasNext() {
        return current.getLeft() != list.getHead();
    }

    public Object next() {
        if (hasNext()) {
            current = current.getLeft();
            return current.getData();
        }
        throw new NoSuchElementException();
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }

    public void reset() {
        current = list.getTail();
    }
}

```

Тестване:

```

package Test;

import java.util.Iterator;
import List.DoublyLinkedList;

public class DoublyLinkedListTester {

```

```

@SuppressWarnings("unchecked")
public void test() {
    DoublyLinkedList list = new DoublyLinkedList();
    int i;
    System.out.println("Testing...");
    System.out.println("      Inserting head:");
    for (i = 0; i < 5; i++) {
        Integer n = new Integer((int) (Math.random() * 99));
        list.addFirst(n);
        System.out.print(n + " ");
    }

    System.out.println();
    System.out.println("      Inserting tail:");
    for (i = 0; i < 5; i++) {
        Integer n = new Integer((int) (Math.random() * 99));
        list.addLast(n);
        System.out.print(n + " ");
    }

    System.out.println();
    System.out.println("      Insert -1 at position 0");
    list.add(0, new Integer(-1));
    System.out.println("      Insert -1 at position " + list.size());
    list.add(list.size(), new Integer(-1));
    System.out.println("      Insert -1 at position " + list.size() / 2);
    list.add(list.size() / 2, new Integer(-1));

    System.out.println("      Iterator: head to tail print...");
    Iterator it = list.iterator();
    while (it.hasNext())
        System.out.print(((Integer) it.next()) + " ");

    System.out.println();
    System.out.print("      Remove element at position " + list.size() / 2
        + ": ");
    System.out.println((Integer) list.remove(list.size() / 2));
    System.out.print("      Remove element at position 0" + ": ");
    System.out.println((Integer) list.remove(0));
    System.out.print("      Remove element at position " + (list.size() - 1)
        + ": ");
    System.out.println((Integer) list.remove(list.size() - 1));

    System.out.println();
    System.out.println("      Removing head:");
    for (i = 0; i < 5; i++) {
        Integer n = (Integer) list.removeFirst();
        System.out.print(n + " ");
    }

    System.out.println();
    System.out.println("      ReverseIterator: tail to head print...");
    it = list.reverseIterator();
    while (it.hasNext())
        System.out.print(((Integer) it.next()) + " ");

    System.out.println();
    System.out.println("      Removing tail:");
    while (!list.isEmpty()) {
        Integer n = (Integer) list.removeLast();
        System.out.print(n + " ");
    }

    System.out.println();
    System.out.println("Done!");
}

```

```

public static void main(String[] args) {
    new DoublyLinkedListTester().test();
}
}

```

Резултати от тестването:

```

Testing...
    Inserting head:
85 58 48 25 72
    Inserting tail:
12 83 79 84 21
        Insert -1 at position 0
        Insert -1 at position 11
        Insert -1 at position 6
        Iterator: head to tail print...
-1 72 25 48 58 85 -1 12 83 79 84 21 -1
        Remove element at position 6: -1
        Remove element at position 0: -1
        Remove element at position 10: -1

    Removing head:
72 25 48 58 85
    ReverseIterator: tail to head print...
21 84 79 83 12
    Removing tail:
21 84 79 83 12
Done!

```

2. Клас **FancyDoublyLinkedList**, съгласно спецификацията:

```

public class FancyDoublyLinkedList extends DoublyLinkedList implements List, FancyList, FancyCollection {
    //Конструктор
    public FancyDoublyLinkedList() {...} //Създава празен списък

    //Наследяване на методите на интерфейса List, реализирани в класа DoublyLinkedList
    //Реализация на методите на интерфейса FancyList
    // Реализация на методите на интерфейса FancyCollection
}

```

Реализация:

```

package List;

import java.util.Iterator;
import Interface.FancyCollection;
import Interface.FancyList;
import Interface.List;

public class FancyDoublyLinkedList extends DoublyLinkedList implements List,
    FancyList, FancyCollection {
    // Constructor
    public FancyDoublyLinkedList() {
        super();
    }

    // List methods - inherited from class DoublyLinkedList

    // FancyList methods implementation
    public boolean containsAll(FancyList c) {
        throw new UnsupportedOperationException();
    }
}

```

```

public boolean addAll(FancyList c) {
    return addAll(size(), c);
}

@SuppressWarnings("unchecked")
public boolean addAll(int index, FancyList c) {
    if (index < 0 || index > size())
        throw new IndexOutOfBoundsException();
    int s = size();
    Iterator it = c.iterator();
    while (it.hasNext())
        add(index++, it.next());
    return s < size();
}

@SuppressWarnings("unchecked")
public boolean removeAll(FancyList c) {
    int s = size();
    Iterator it = c.iterator();
    while (it.hasNext()) {
        Object temp = it.next();
        boolean flag = true;
        while (flag) {
            flag = false;
            if (remove(temp) != null)
                flag = true;
        }
    }
    return s > size();
}

@SuppressWarnings("unchecked")
public boolean retainAll(FancyList c) {
    int s = size();
    Iterator it = this.iterator();
    while (it.hasNext()) {
        Object temp = it.next();
        if (!c.contains(temp))
            remove(temp);
    }
    return s > size();
}

public FancyList subList(int fromIndex, int toIndex) {
    if (fromIndex < 0 || toIndex > size() || fromIndex > toIndex)
        throw new IndexOutOfBoundsException();
    FancyDoublyLinkedList result = new FancyDoublyLinkedList();
    for (int i = fromIndex; i < toIndex; i++)
        result.add(get(i));
    return result;
}

// FancyCollection methods implementation
public void clear() {
    removeAll(this);
}

public Object[] toArray() {
    throw new UnsupportedOperationException();
}

public Object clone() {
    throw new UnsupportedOperationException();
}

public boolean equals(Object arg) {

```

```

        throw new UnsupportedOperationException();
    }
}

```

Тестване:

```

package Test;

import List.FancyDoublyLinkedList;

public class FancyDoublyLinkedListTester {
    public void test() {
        FancyDoublyLinkedList A = new FancyDoublyLinkedList();
        for (int i = 10; i <= 20; i++)
            A.add(new Integer(i));
        System.out.println("      FancyDoublyLinkedList A:");
        System.out.println(A);

        FancyDoublyLinkedList B = new FancyDoublyLinkedList();
        for (int i = 15; i <= 25; i++)
            B.add(new Integer(i));
        System.out.println("      FancyDoublyLinkedList B:");
        System.out.println(B);

        A.addAll(3, B);
        System.out.println("      FancyDoublyLinkedList A = A.addAll(3,B):");
        System.out.println(A);

        A.removeAll(B);
        System.out.println("      FancyDoublyLinkedList A = A.removeAll(B):");
        System.out.println(A);

        FancyDoublyLinkedList C = (FancyDoublyLinkedList) A.subList(1, 3);
        System.out.println("      FancyDoublyLinkedList C = A.subList(1,3):");
        System.out.println(C);

        A.retainAll(C);
        System.out.println("      FancyDoublyLinkedList A = A.retainAll(C):");
        System.out.println(A);
    }

    public static void main(String[] args) {
        new FancyDoublyLinkedListTester().test();
    }
}

```

Резултати от тестването:

```

FancyDoublyLinkedList A:
[10 11 12 13 14 15 16 17 18 19 20 ]
FancyDoublyLinkedList B:
[15 16 17 18 19 20 21 22 23 24 25 ]
FancyDoublyLinkedList A = A.addAll(3,B):
[10 11 12 15 16 17 18 19 20 21 22 23 24 25 13 14 15 16 17 18 19 20 ]
FancyDoublyLinkedList A = A.removeAll(B):
[10 11 12 13 14 ]
FancyDoublyLinkedList C = A.subList(1,3):
[11 12 ]
FancyDoublyLinkedList A = A.retainAll(C):
[11 12 ]

```

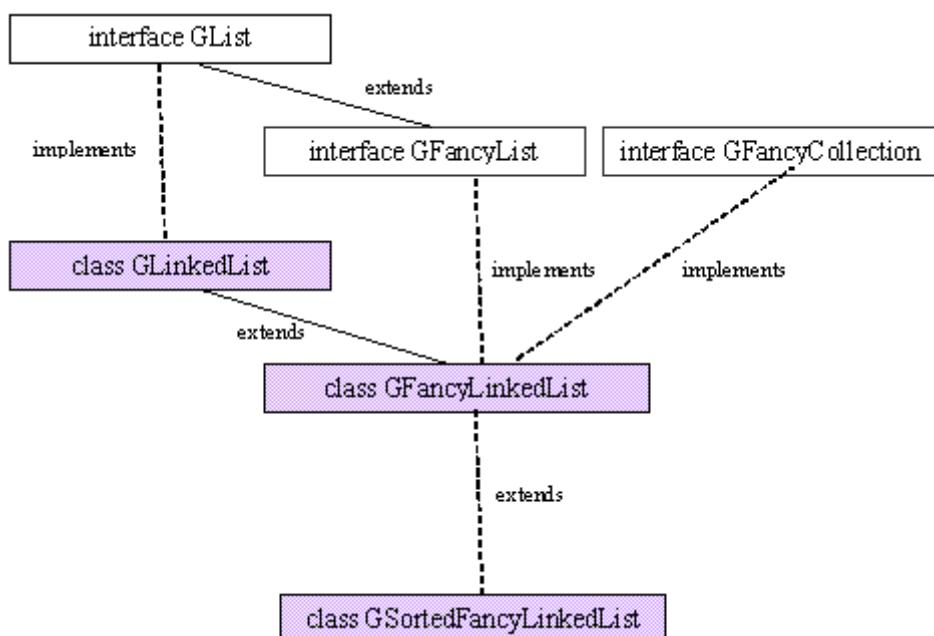


Задачи за упражнение

Задача: Да се реализира линеен двусвързан списък с възможност за “заключване” на негови възли. “Заключен” възел не може да бъде изключен от списъка, на който принадлежи. Реализацията да включва:

1. Клас **LockDNode** -наследник на **DNode**, съдържащ променлива за състоянието на текущия възел –заключен или не, конструктор с параметър за данните, като при създаването на възел той не е заключен, метод за заключване на текущия възел и метод, който проверява дали текущия възел е заключен
2. Клас **LockedDoublyLinkedList** - наследник на **DoublyLinkedList**, съдържащ допълнителен метод void lockNode(LockDNode node) за заключване на възела node. В класа е необходимо да се предефинират методите за включване и изключване

Задача: Да се реализира йерархията от родови интерфейси и класове за свързан списък:

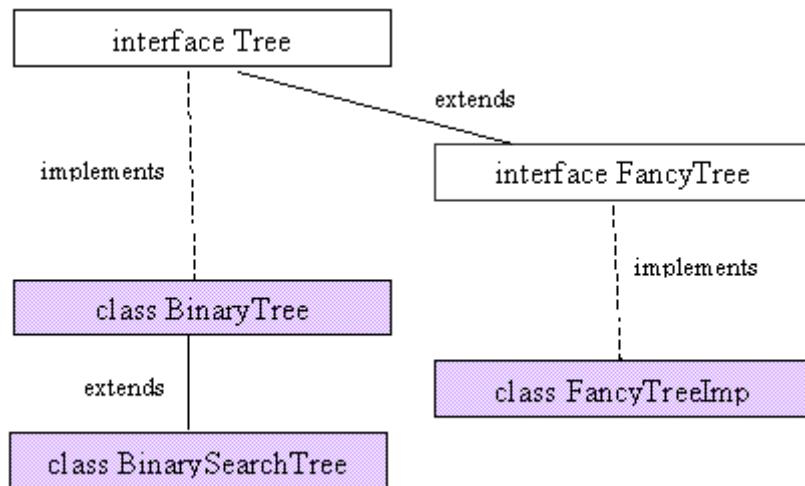


Интерфейсите **GList** и **GFancyList** са родов вариант на интерфейсите **List** и **FancyList**.



Двойно дърво

Задача: Дадена е юерархията от интерфейси и класове:



Интерфейсът **Tree** представя операции на *дърво с елементи от тип Object*:

```

package Interface;

import java.util.Iterator;

public interface Tree {
    public void add(Object element);

    public Object remove(Object element);

    public boolean contains(Object element);

    public boolean isEmpty();

    public int size();

    public int height();

    public int numberOfLeaves();

    public void clear();

    @SuppressWarnings("unchecked")
    public Iterator iterator();

    @SuppressWarnings("unchecked")
    public Iterator treeInorderIterator();

    @SuppressWarnings("unchecked")
    public Iterator treePreorderIterator();

    @SuppressWarnings("unchecked")
    public Iterator treePostorderIterator();

    @SuppressWarnings("unchecked")
    public Iterator treeBreadthFirstIterator();

    @SuppressWarnings("unchecked")
    public Iterator subtreesIterator();

    public Object clone();
}
  
```

```
public String toString();
}
```

Интерфейсът **FancyTree** представя операции над дървета:

```
package Interface;

public interface FancyTree extends Tree {
    public boolean sameElements(FancyTree arg);

    public boolean sameStructure(FancyTree arg);

    public boolean sameTrees(FancyTree arg);

    public boolean sameHeight(FancyTree arg);

    public boolean sameNumberOfLeaves(FancyTree arg);

    public boolean containsAll(FancyTree arg);

    public boolean addAll(FancyTree arg);

    public boolean removeAll(FancyTree arg);

    public boolean retainAll(FancyTree arg);
}
```

Класът **TreeNode** представя възел на двоично дърво с елементи от тип **Object**:

```
package Tree;

public class TreeNode {
    // Data
    protected Object data;
    protected TreeNode left, right;

    // Constructor
    public TreeNode(Object d) {
        data = d;
        left = right = null;
    }

    // Methods
    public Object data() {
        return data;
    }

    public Object setData(Object d) {
        return data = d;
    }

    public TreeNode left() {
        return left;
    }

    public TreeNode setLeft(TreeNode n) {
        return left = n;
    }

    public TreeNode right() {
        return right;
    }

    public TreeNode setRight(TreeNode n) {
```

```

        return right = n;
    }
}

```

Да се реализират:

1. Клас **BinaryTree** – двоично дърво с елементи от тип **Object**, съгласно спецификацията:

```

package Tree;

import java.util.Iterator;
import Interface.Tree;
import Iterator.BTreeBreadthFirstIterator;
import Iterator.BTreeInorderIterator;
import Iterator.BTreePostorderIterator;
import Iterator.BTreePreorderIterator;
import Iterator.GArrayIterator;

public class BinaryTree implements Tree {
    //Данни
    protected TreeNode root;

    //Конструктори
    public BinaryTree() { this(null); }
    public BinaryTree(TreeNode n) { root = n; }

    //Методи за достъп
    public TreeNode getRoot() { return root; }
    public BinaryTree getLeftSubtree() { return isEmpty() ? null : new BinaryTree(root.left); }
    public BinaryTree getRightSubtree() { return isEmpty() ? null : new BinaryTree(root.right); }

    //Реализация на методите на интерфейса Tree
}

```

Реализация:

```

package Tree;

import java.util.Iterator;
import Interface.Tree;
import Iterator.BTreeBreadthFirstIterator;
import Iterator.BTreeInorderIterator;
import Iterator.BTreePostorderIterator;
import Iterator.BTreePreorderIterator;
import Iterator.GArrayIterator;

public class BinaryTree implements Tree {
    // Data
    protected TreeNode root;

    // Constructors
    public BinaryTree() {
        this(null);
    }

    public BinaryTree(TreeNode n) {
        root = n;
    }

    // Access methods
    public TreeNode getRoot() {
        return root;
    }
}

```

```

public BinaryTree getLeftSubtree() {
    return isEmpty() ? null : new BinaryTree(root.left);
}

public BinaryTree getRightSubtree() {
    return isEmpty() ? null : new BinaryTree(root.right);
}

// Tree methods
public void add(Object element) {
    if (isEmpty())
        root = new TreeNode(element);
    else {
        int w = (int) (100.0 * Math.random()) % 2;
        if (w == 0)
            if (root.left == null)
                root.left = new TreeNode(element);
            else
                getLeftSubtree().add(element);
        else if (root.right == null)
            root.right = new TreeNode(element);
        else
            getRightSubtree().add(element);
    }
}

public Object remove(Object element) {
    throw new UnsupportedOperationException();
}

public boolean contains(Object element) {
    if (isEmpty())
        return false;
    if (root.data.equals(element))
        return true;
    return getLeftSubtree().contains(element) ? true : getRightSubtree()
        .contains(element);
}

public boolean isEmpty() {
    return root == null;
}

public int size() {
    return isEmpty() ? 0 : getLeftSubtree().size() + 1
        + getRightSubtree().size();
}

public int height() {
    return isEmpty() ? -1 : Math.max(getLeftSubtree().height(),
        getRightSubtree().height()) + 1;
}

public int numberofLiefs() {
    if (isEmpty())
        return 0;
    BinaryTree leftTree = getLeftSubtree();
    BinaryTree rightTree = getRightSubtree();
    return leftTree.isEmpty() && rightTree.isEmpty() ? 1 : leftTree
        .numberofLiefs()
        + rightTree.numberofLiefs();
}

public void clear() {
    root = null;
}

```

```

@SuppressWarnings("unchecked")
public Iterator iterator() {
    return new BTreeInorderIterator(this);
}

@SuppressWarnings("unchecked")
public Iterator treeInorderIterator() {
    return new BTreeInorderIterator(this);
}

@SuppressWarnings("unchecked")
public Iterator treePreorderIterator() {
    return new BTreePreorderIterator(this);
}

@SuppressWarnings("unchecked")
public Iterator treePostorderIterator() {
    return new BTreePostorderIterator(this);
}

@SuppressWarnings("unchecked")
public Iterator treeBreadthFirstIterator() {
    return new BTreeBreadthFirstIterator(this);
}

@SuppressWarnings("unchecked")
public Iterator subtreesIterator() {
    BinaryTree[] temp = { getLeftSubtree(), getRightSubtree() };
    return (Iterator) (new GArrayIterator<BinaryTree>(temp));
}

public Object clone() {
    // return new BinaryTree(this.copy(new BinaryTree()));
    if (!this.isEmpty()) {
        BinaryTree tree = new BinaryTree(new TreeNode(this.root.data));
        tree.root.left = ((BinaryTree) (this.getLeftSubtree().clone())).root;
        tree.root.right = ((BinaryTree)
(this.getRightSubtree().clone())).root;
        return tree;
    }
    return new BinaryTree();
}

public String toString() {
    if (!this.isEmpty()) {
        String str = "\n" + this.root.data() + ": ";
        str = str + (this.root.left != null ? this.root.left.data : "null")
            + ", ";
        str = str
            + (this.root.right != null ? this.root.right.data : "null");
        str = str + this.getLeftSubtree().toString();
        str = str + this.getRightSubtree().toString();
        return str;
    }
    return "";
}

```

Класът **BTreeInorderIterator** представя итератор за *смесено обхождане на двоично дърво с елементи от тип Object*:

package Iterator;

```

import java.util.Iterator;
import java.util.LinkedList;

import Tree.BinaryTree;

@SuppressWarnings("unchecked")
public class BTreInorderIterator implements Iterator {
    // Data
    LinkedList list;
    Iterator it;

    // Constructor
    public BTreInorderIterator(BinaryTree tree) {
        list = new LinkedList();
        inorder(tree);
        reset();
    }

    // Private method
    private void inorder(BinaryTree tree) {
        if (!tree.isEmpty()) {
            inorder(tree.getLeftSubtree());
            list.addLast(tree.getRoot().data());
            inorder(tree.getRightSubtree());
        }
    }

    // Iterator methods
    public boolean hasNext() {
        return it.hasNext();
    }

    public Object next() {
        return it.next();
    }

    public void reset() {
        it = list.iterator();
    }

    public void remove() {
        it.remove();
    }
}

```

Класът **BTrePreorderIterator** представя итератор за *назходящо обхождане на двоично дърво с елементи от тип Object*:

```

package Iterator;

import java.util.Iterator;
import java.util.LinkedList;
import Tree.BinaryTree;

@SuppressWarnings("unchecked")
public class BTrePreorderIterator implements Iterator {
    // Data
    LinkedList list;
    Iterator it;

    // Constructor
    public BTrePreorderIterator(BinaryTree tree) {
        list = new LinkedList();
        preorder(tree);
        reset();
    }

    // Private method
    private void preorder(BinaryTree tree) {
        list.addLast(tree.getRoot().data());
        if (!tree.isEmpty()) {
            preorder(tree.getLeftSubtree());
            preorder(tree.getRightSubtree());
        }
    }

    // Iterator methods
    public boolean hasNext() {
        return it.hasNext();
    }

    public Object next() {
        return it.next();
    }

    public void reset() {
        it = list.iterator();
    }

    public void remove() {
        it.remove();
    }
}

```

```

}

// Private method
private void preorder(BinaryTree tree) {
    if (!tree.isEmpty()) {
        list.addLast(tree.getRoot().data());
        preorder(tree.getLeftSubtree());
        preorder(tree.getRightSubtree());
    }
}

// Iterator methods
public boolean hasNext() {
    return it.hasNext();
}

public Object next() {
    return it.next();
}

public void reset() {
    it = list.iterator();
}

public void remove() {
    it.remove();
}
}

```

Класът **BTreePostorderIterator** представя итератор за възходящо обхождане на двоично дърво с елементи от тип *Object*:

```

package Iterator;

import java.util.Iterator;
import java.util.LinkedList;

import Tree.BinaryTree;

@SuppressWarnings("unchecked")
public class BTreePostorderIterator implements Iterator {
    // Data
    LinkedList list;
    Iterator it;

    // Constructor
    public BTreePostorderIterator(BinaryTree tree) {
        list = new LinkedList();
        postorder(tree);
        reset();
    }

    // Private method
    private void postorder(BinaryTree tree) {
        if (!tree.isEmpty()) {
            postorder(tree.getLeftSubtree());
            postorder(tree.getRightSubtree());
            list.addLast(tree.getRoot().data());
        }
    }

    // Iterator methods
    public boolean hasNext() {
        return it.hasNext();
    }
}

```

```

public Object next() {
    return it.next();
}

public void reset() {
    it = list.iterator();
}

public void remove() {
    it.remove();
}
}

```

Класът **BTreeBreadthFirstIterator** представя итератор за *обхождане по нива на двоично дърво с елементи от тип Object*, с възможност за обработка на данните от дадено ниво:

```

package Iterator;

import java.util.Iterator;
import java.util.LinkedList;
import java.util.NoSuchElementException;
import java.util.Vector;

import Tree.BinaryTree;
import TreeTreeNode;

@SuppressWarnings("unchecked")
public class BTreeBreadthFirstIterator implements Iterator {
    // Data
    Vector<LinkedList> levels; // Vector of levels
    Iterator it;
    int level;

    // Constructor
    public BTreeBreadthFirstIterator(BinaryTree tree) {
        levels = new Vector<LinkedList>();
        breadthFirstTraversal(tree);
        reset();
    }

    // Private method
    private void breadthFirstTraversal(BinaryTree tree) {
        LinkedList<TreeNode> currentLevel = new LinkedList<TreeNode>();
        if (!tree.isEmpty())
            currentLevel.addLast(tree.getRoot());
        while (!currentLevel.isEmpty()) {
            LinkedList data = new LinkedList();
            LinkedList<TreeNode> nextLevel = new LinkedList<TreeNode>();
            Iterator<TreeNode> it = currentLevel.iterator();
            while (it.hasNext()) {
                TreeNode n = it.next();
                data.addLast(n.data());
                if (n.left() != null)
                    nextLevel.addLast(n.left());
                if (n.right() != null)
                    nextLevel.addLast(n.right());
            }
            levels.add(data);
            level++;
            currentLevel = nextLevel;
        }
    }

    // Methods
}

```

```

public int numberOfLevels() {
    return levels.size();
}

public LinkedList getData(int level) {
    return levels.get(level);
}

// Iterator methods
public boolean hasNext() {
    if (it.hasNext())
        return true;
    level++;
    if (level < levels.size()) {
        it = levels.get(level).iterator();
        return true;
    }
    return false;
}

public Object next() {
    if (!hasNext())
        throw new NoSuchElementException();
    return it.next();
}

public void remove() {
    throw new UnsupportedOperationException();
}

public void reset() {
    level = 0;
    it = levels.get(0).iterator();
}
}

```

Класът **BTreeBreadthFirstIteratorEasy** представя итератор за обхождане по нива на двоично дърво с елементи от тип *Object*, без отделяне на нивата:

```

package Iterator;

import java.util.Iterator;
import java.util.LinkedList;

import Tree.BinaryTree;
import TreeTreeNode;

@SuppressWarnings("unchecked")
public class BTreeBreadthFirstIteratorEasy implements Iterator {
    // Data
    LinkedList list;
    Iterator it;

    // Constructor
    public BTreeBreadthFirstIteratorEasy(BinaryTree tree) {
        list = new LinkedList();
        breadthFirstTraversal(tree);
        reset();
    }

    // Private method
    private void breadthFirstTraversal(BinaryTree tree) {
        LinkedList<TreeNode> queue = new LinkedList<TreeNode>();
        queue.addLast(tree.getRoot());
        while (!queue.isEmpty()) {
            TreeNode node = queue.removeFirst();
            list.add(node);
            if (node.left != null)
                queue.addLast(node.left);
            if (node.right != null)
                queue.addLast(node.right);
        }
    }
}

```

```

TreeNode temp = queue.removeFirst();
list.addLast(temp.data());
if (temp.left() != null)
    queue.addLast(temp.left());
if (temp.right() != null)
    queue.addLast(temp.right());
}
}

// Iterator methods
public boolean hasNext() {
    return it.hasNext();
}

public Object next() {
    return it.next();
}

public void reset() {
    it = list.iterator();
}

public void remove() {
    it.remove();
}
}

```

Тестване: Програмата **BinaryTreeTester**:

- Чете думи от файла testTreeData1.txt и създава двоично дърво от думи – обекти на класа **String**

BinaryTree tree = new BinaryTree()

Обхожда дървото по четирите начина, като записва резултата във файла testTreeResult1.txt.

- Чете низове от вида: дума@цяло_число от файла testTreeData2.txt и създава двоично дърво от двойки (дума,цяло_число) – обекти на класа **Entry**

tree = new BinaryTree()

Обхожда дървото по четирите начина, като записва резултата във файла testTreeResult2.txt.

- Чете низове от вида: дума@цяло_число от файла testTreeData3.txt и създава двоично дърво от двойки (дума,цяло_число) – обекти на класа **Entry**

tree = new BinaryTree()

Обхожда дървото по четирите начина, като записва резултата във файла testTreeResult3.txt.

```

package Test;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Iterator;
import Function.Entry;
import Iterator.BTreeBreadthFirstIterator;
import Tree.BinaryTree;

public class BinaryTreeTester {
    public void createTreeOfStrings(BinaryTree tree, String source)
        throws IOException {
        BufferedReader in = new BufferedReader(new FileReader(source));
        for (int i = 1; i <= 10; i++)
            tree.add(in.readLine());

        in.close();
    }
}

```

```

}

public void createTreeOfEntries(BinaryTree tree, String source)
    throws IOException {
    BufferedReader in = new BufferedReader(new FileReader(source));

    for (int i = 1; i <= 10; i++) {
        String line = in.readLine();
        int index = line.indexOf('@');
        String word = line.substring(0, index);
        String digit = line.substring(index + 1);
        Entry m = new Entry(word);
        m.setValue(Integer.valueOf(digit));
        tree.add(m);
    }

    in.close();
}

@SuppressWarnings("unchecked")
public void test(BinaryTree tree, String target) throws IOException {
    PrintWriter out;
    out = new PrintWriter(new BufferedWriter(new FileWriter(target)));

    out.println("      Testing... ");
    out.println("Height of the tree: " + tree.height());

    out.println("      Tree breadth-first traversal:");
    Iterator it = tree.treeBreadthFirstIterator();
    BTreBreadthFirstIterator itw = (BTreBreadthFirstIterator) it;
    for (int i = 0; i < itw.numberofLevels(); i++) {
        java.util.Iterator currentLevelIt = itw.getData(i).iterator();
        out.println("      Level #: " + i);
        while (currentLevelIt.hasNext())
            out.println(currentLevelIt.next());
    }

    out.println("      Tree inorder traversal:");
    it = tree.treeInorderIterator();
    while (it.hasNext())
        out.println(it.next());

    out.println("      Tree preorder traversal:");
    it = tree.treePreorderIterator();
    while (it.hasNext())
        out.println(it.next());

    out.println("      Tree postorder traversal:");
    it = tree.treePostorderIterator();
    while (it.hasNext())
        out.println(it.next());

    out.println("Done!");
    out.close();
}

public void test() {
    try {
        String s = File.separator;
        String userDir = System.getProperty("user.dir");
        System.out.println("User directory: " + userDir);
        String dir = userDir + s + "src" + s + "Test";

        String source = dir + s + "testTreeData1.txt";
        System.out.println("Read text from the file " + source);
        String target = dir + s + "testTreeResult1.txt";
    }
}

```

```

System.out.println("Test and write results to the file " + target);
BinaryTree tree = new BinaryTree();
createTreeOfStrings(tree, source);
test(tree, target);

source = dir + s + "testTreeData2.txt";
System.out.println("Read text from the file " + source);
target = dir + s + "testTreeResult2.txt";
System.out.println("Test and write results to the file " + target);
tree = new BinaryTree();
createTreeOfEntries(tree, source);
test(tree, target);

source = dir + s + "testTreeData3.txt";
System.out.println("Read text from the file " + source);
target = dir + s + "testTreeResult3.txt";
System.out.println("Test and write results to the file " + target);
tree = new BinaryTree();
createTreeOfEntries(tree, source);
test(tree, target);

} catch (IOException e) {
}
}

public static void main(String[] args) {
    new BinaryTreeTester().test();
}
}

```

Класът **Entry** представя двойка от вида (ключ, стойност):

```

package Function;

public class Entry {
    // Data
    private Object key;
    private Object value;

    // Constructor
    public Entry(Object key) throws NullPointerException {
        if (key == null)
            throw new NullPointerException();
        this.key = key;
    }

    // Methods
    public Object getKey() {
        return key;
    }

    public Object getValue() {
        return value;
    }

    public Object setValue(Object newValue) throws NullPointerException {
        if (newValue == null)
            throw new NullPointerException();
        Object old = value;
        value = newValue;
        return old;
    }

    public boolean equals(Object obj) {
        Entry t = (Entry) obj;
        return key.equals(t.getKey()) && value.equals(t.getValue());
    }
}

```

```

    }

    public String toString() {
        return "(" + key + ", " + value + ")";
    }
}

```

2. Клас **BinarySearchTree** – двоично наредено дърво с елементи от тип **Object**, съгласно спецификацията:

```

package Tree;

import java.util.Comparator;
import Comparator.NaturalComparator;
import Interface.Tree;

public class BinarySearchTree extends BinaryTree implements Tree {
    //Данни
    protected Comparator c;

    //Конструктори
    public BinarySearchTree() { this(new NaturalComparator()); }
    public BinarySearchTree(Comparator c) { this(c,null); }
    public BinarySearchTree(Comparator c,TreeNode n) { super(n); this.c = c; }

    //Методи за достъп – наследени от класа BinaryTree
    public Comparator getComparator() { return c; }

    //Предефиниране на методи за достъп, наследени от класа BinaryTree
    public BinaryTree getLeftSubtree() { return isEmpty() ? null : new BinarySearchTree(c.root.left); }
    public BinaryTree getRightSubtree() { return isEmpty() ? null : new BinarySearchTree(c.root.right); }

    //Наследяване на методите на интерфейса Tree, реализирани в класа BinaryTree

    //Предефиниране на методи, наследени от класа BinaryTree
    public void add(Object element) {...}
    public Object remove(Object element) {...}
    public boolean contains(Object element) {...}
}

```

Реализация:

```

package Tree;

import java.util.Comparator;
import Comparator.NaturalComparator;
import Interface.Tree;

public class BinarySearchTree extends BinaryTree implements Tree {
    // Data
    @SuppressWarnings("unchecked")
    protected Comparator c;

    // Constructors
    @SuppressWarnings("unchecked")
    public BinarySearchTree() {
        this(new NaturalComparator());
    }

    @SuppressWarnings("unchecked")
    public BinarySearchTree(Comparator c) {
        this(c, null);
    }
}

```

```

@SuppressWarnings("unchecked")
public BinarySearchTree(Comparator c, TreeNode n) {
    super(n);
    this.c = c;
}

// Access methods - inherited from class BinaryTree
@SuppressWarnings("unchecked")
public Comparator getComparator() {
    return c;
}

// Override methods
public BinaryTree getLeftSubtree() {
    return isEmpty() ? null : new BinarySearchTree(c, root.left);
}

public BinaryTree getRightSubtree() {
    return isEmpty() ? null : new BinarySearchTree(c, root.right);
}

// Private method
@SuppressWarnings("unchecked")
private TreeNode remove(TreeNode r, Object n) {
    if (r != null) {
        if (c.compare(r.data, n) < 0) {
            r.right = remove(r.right, n);
        } else if (c.compare(r.data, n) > 0) {
            r.left = remove(r.left, n);
        } else { // c.compare(r.data,n) == 0
            if (r.left == null && r.right == null) {
                r = null;
            } else if (r.left != null && r.right == null) {
                r = r.left;
            } else if (r.right != null && r.left == null) {
                r = r.right;
            } else {
                if (r.right.left == null) {
                    r.right.left = r.left;
                    r = r.right;
                } else {
                    TreeNode q, p = r.right;
                    while (p.left.left != null)
                        p = p.left;
                    q = p.left;
                    p.left = q.right;
                    q.left = r.left;
                    q.right = r.right;
                    r = q;
                }
            }
        }
    }
    return r;
}

// Tree methods - inherited from class BinaryTree

// Override methods
@SuppressWarnings("unchecked")
public void add(Object element) {
    if (isEmpty())
        root = new TreeNode(element);
    else if (c.compare(element, root.data) <= 0)
        if (root.left == null)
            root.left = new TreeNode(element);
}

```

```

        else
            getLeftSubtree().add(element);
        else if (root.right == null)
            root.right = new TreeNode(element);
        else
            getRightSubtree().add(element);
    }

public Object remove(Object element) {
    TreeNode r = remove(root, element);
    return r != null ? r.data : null;
}

@SuppressWarnings("unchecked")
public boolean contains(Object element) {
    if (isEmpty())
        return false;
    if (c.compare(element, root.data) == 0)
        return true;
    else if (c.compare(element, root.data) < 0)
        return getLeftSubtree().contains(element);
    else
        return getRightSubtree().contains(element);
}
}

```

Тестване: Програмата **BinarySearchTreeTester**:

1. Чете думи от файла testTreeData1.txt и създава двоично наредено дърво от думи – обекти на класа **String**, като се използва **NaturalComparator<E>**

BinaryTree tree = new BinarySearchTree(new NaturalComparator<String>())

Обхожда дървото по четирите начина, като записва резултата във файла testTreeResult1.txt.

1. Чете низове от вида: дума@цяло_число от файла testTreeData2.txt и създава двоично наредено дърво от двойки (дума,цяло_число) – обекти на класа **Entry**, като се използва **EntryComparator**

tree = new BinarySearchTree(new EntryComparator())

Обхожда дървото по четирите начина, като записва резултата във файла testTreeResult2.txt.

2. Чете низове от вида: дума@цяло_число от файла testTreeData3.txt и създава двоично дърво от двойки (дума,цяло_число) – обекти на класа **Entry**, като се използва **EntryComparator2**

tree = new BinarySearchTree(new EntryComparator2())

Обхожда дървото по четирите начина, като записва резултата във файла testTreeResult3.txt.

```

package Test;

import java.io.File;
import java.io.IOException;
import Comparator.EntryComparator;
import Comparator.EntryComparator2;
import Comparator.NaturalComparator;
import Tree.BinarySearchTree;
import Tree.BinaryTree;

public class BinarySearchTreeTester extends BinaryTreeTester {
    public void test() {
        try {
            String s = File.separator;
            String userDir = System.getProperty("user.dir");
            System.out.println("User directory: " + userDir);
            String dir = userDir + s + "src" + s + "Test";

            String source = dir + s + "testTreeData1.txt";
            System.out.println("Read text from the file " + source);
            String target = dir + s + "testTreeResult1.txt";

```

```

System.out.println("Test and write results to the file " + target);
BinaryTree tree = new BinarySearchTree(
    new NaturalComparator<String>());
super.createTreeOfStrings(tree, source);
super.test(tree, target);

source = dir + s + "testTreeData2.txt";
System.out.println("Read text from the file " + source);
target = dir + s + "testTreeResult2.txt";
System.out.println("Test and write results to the file " + target);
tree = new BinarySearchTree(new EntryComparator());
super.createTreeOfEntries(tree, source);
super.test(tree, target);
System.out.println("Tree" + tree);

target = dir + s + "testTreeCopyResult2.txt";
System.out.println("Test and write results to the file " + target);
BinaryTree copy = (BinaryTree) tree.clone();
super.test(copy, target);
System.out.println("Tree copy" + copy);
source = dir + s + "testTreeData3.txt";
System.out.println("Read text from the file " + source);
target = dir + s + "testTreeResult3.txt";
System.out.println("Test and write results to the file " + target);
tree = new BinarySearchTree(new EntryComparator2());
super.createTreeOfEntries(tree, source);
super.test(tree, target);

} catch (IOException e) {
}
}

public static void main(String[] args) {
    new BinarySearchTreeTester().test();
}
}

```

Класът **EntryComparator** представя обект за сравняване на ключовете на два обекта на класа **Entry**:

```

package Comparator;

import java.util.Comparator;
import Function.Entry;

public class EntryComparator implements Comparator<Entry> {
    @SuppressWarnings("unchecked")
    public int compare(Entry a, Entry b) {
        Object key1 = a.getKey();
        Object key2 = b.getKey();
        return ((Comparable) key1).compareTo(key2);
    }
}

```

Класът **EntryComparator2** представя обект за сравняване на стойностите на два обекта на класа **Entry**:

```

package Comparator;

import java.util.Comparator;
import Function.Entry;

public class EntryComparator2 implements Comparator<Entry> {
    @SuppressWarnings("unchecked")
    public int compare(Entry a, Entry b) {

```

```

        Object value1 = a.getValue();
        Object value2 = b.getValue();
        return ((Comparable) value1).compareTo(value2);
    }
}

```

Резултати от тестването:

```

User directory: D:\UserFolder\JavaProjects\SDP
Read text from the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTreeData1.txt
Test and write results to the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTreeResult1.txt
Read text from the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTreeData2.txt
Test and write results to the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTreeResult2.txt
Tree
(zero,0): (one,1), null
(one,1): (four,4), (two,2)
(four,4): (five,5), (nine,9)
(five,5): (eight,8), null
(eight,8): null, null
(nine,9): null, null
(two,2): (three,3), null
(three,3): (six,6), null
(six,6): (seven,7), null
(seven,7): null, null
Test and write results to the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTreeCopyResult2.txt
Tree copy
(zero,0): (one,1), null
(one,1): (four,4), (two,2)
(four,4): (five,5), (nine,9)
(five,5): (eight,8), null
(eight,8): null, null
(nine,9): null, null
(two,2): (three,3), null
(three,3): (six,6), null
(six,6): (seven,7), null
(seven,7): null, null
Read text from the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTreeData3.txt
Test and write results to the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTreeResult3.txt

```

Файл testTreeData1.txt

Word2
Word10
Word7
Word1
Word5
Word7
Word2
Word9
Word8
Word7

Файл testTreeResult1.txt

```

Testing...
Height of the tree: 4
      Tree breadth-first traversal:
      Level #: 0
Word2
      Level #: 1
Word10
Word7
      Level #: 2
Word1
Word2
Word5
Word9
      Level #: 3
Word7
Word8
      Level #: 4
Word7

```

```
Tree inorder traversal:  
Word1  
Word10  
Word2  
Word2  
Word5  
Word7  
Word7  
Word7  
Word8  
Word9  
Tree preorder traversal:  
Word2  
Word10  
Word1  
Word2  
Word7  
Word5  
Word7  
Word7  
Word9  
Word8  
Tree postorder traversal:  
Word1  
Word2  
Word10  
Word7  
Word7  
Word5  
Word8  
Word9  
Word7  
Word2  
Done!
```

Файл testTreeData2.txt

```
zero@0  
one@1  
two@2  
three@3  
four@4  
five@5  
six@6  
seven@7  
eight@8  
nine@9
```

Файл testTreeResult2.txt

```
Testing...  
Height of the tree: 5  
Tree breadth-first traversal:  
Level #: 0  
(zero,0)  
Level #: 1  
(one,1)  
Level #: 2  
(four,4)  
(two,2)  
Level #: 3  
(five,5)  
(nine,9)  
(three,3)  
Level #: 4  
(eight,8)  
(six,6)  
Level #: 5  
(seven,7)  
Tree inorder traversal:  
(eight,8)  
(five,5)  
(four,4)  
(nine,9)  
(one,1)  
(seven,7)  
(six,6)  
(three,3)  
(two,2)  
(zero,0)
```

```

        Tree preorder traversal:
(zero,0)
(one,1)
(four,4)
(five,5)
(eight,8)
(nine,9)
(two,2)
(three,3)
(six,6)
(seven,7)
        Tree postorder traversal:
(eight,8)
(five,5)
(nine,9)
(four,4)
(seven,7)
(six,6)
(three,3)
(two,2)
(one,1)
(zero,0)
Done!

```

Файл testTreeData3.txt

```

zero@52
one@64
two@98
three@52
four@12
five@9
six@41
seven@26
eight@8
nine@69

```

Файл testTreeResult3.txt

```

Testing...
Height of the tree: 4
        Tree breadth-first traversal:
        Level #: 0
(zero,52)
        Level #: 1
(three,52)
(one,64)
        Level #: 2
(four,12)
(two,98)
        Level #: 3
(five,9)
(six,41)
(nine,69)
        Level #: 4
(eight,8)
(seven,26)
        Tree inorder traversal:
(eight,8)
(five,9)
(four,12)
(seven,26)
(six,41)
(three,52)
(zero,52)
(one,64)
(nine,69)
(two,98)
        Tree preorder traversal:
(zero,52)
(three,52)
(four,12)
(five,9)
(eight,8)
(six,41)
(seven,26)
(one,64)
(two,98)
(nine,69)

```

```
Tree postorder traversal:  
(eight,8)  
(five,9)  
(seven,26)  
(six,41)  
(four,12)  
(three,52)  
(nine,69)  
(two,98)  
(one,64)  
(zero,52)  
Done!
```

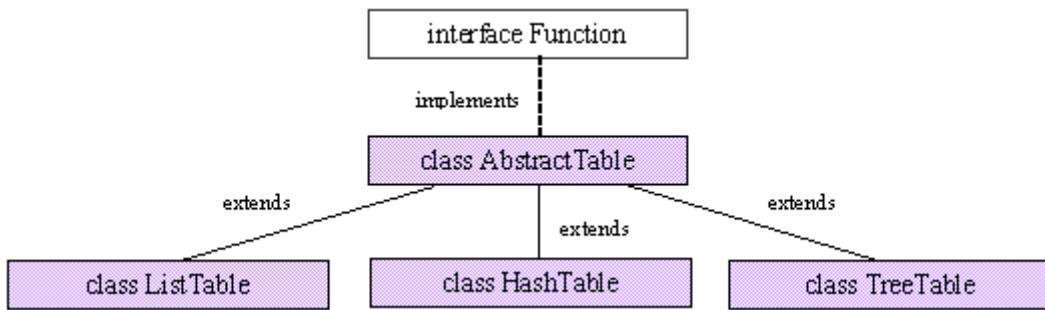
3. Клас **FancyTreeImp**, съгласно спецификацията:

```
package Tree;  
  
import java.util.Iterator;  
import Interface.FancyTree;  
import Interface.Tree;  
  
public class FancyTreeImp implements Tree, FancyTree {  
    //Данни  
    private Tree elements;  
  
    //Конструктор  
    public FancyTreeImp(Tree arg) { elements = arg; }  
  
    //Tree methods implementation  
    //Реализация на методите на интерфейса Tree  
    //Реализация на методите на интерфейса FancyTree  
}
```



Таблица

Задача: Дадена е юерархията от интерфейси и класове:



Интерфейсът **Function** представя операции на *таблица на функция* със стойности на променливата (наричани понататък *ключове*), от тип **Object** и функционални стойности (наричани понататък *стойности*) от тип **Object**:

```

package Interface;

import java.util.Collection;
import java.util.Iterator;
import java.util.Set;

public interface Function {
    public boolean containsKey(Object key);
    //Проверява дали в текущата таблица се съдържа ключът key

    public boolean containsValue(Object value);
    //Проверява дали в текущата таблица се съдържа стойността value

    public Object get(Object key);
    //Връща стойността на функцията за ключа key от текущата таблица

    public Object put(Object key, Object value);
    //Ако има ред с ключ key в текущата таблица, съответната стойност се променя
    //на value. В противен случай се включва нов ред (key,value) в текущата
    //таблица

    public Object remove(Object key);
    //Изключва ред с ключ key от текущата таблица и връща съответната стойност

    public boolean isEmpty();
    //Проверява дали текущата таблица е празна

    public boolean equals(Object obj);
    //Проверява дали текущата таблица и таблицата obj представляват една и съща
    //функция

    public int size();
    //Връща броя на редовете на текущата таблица

    @SuppressWarnings("unchecked")
    public Iterator keys();
    //Обхожда ключовете на текущата таблица

    @SuppressWarnings("unchecked")
    public Iterator values();
    //Обхожда стойностите на текущата таблица

    @SuppressWarnings("unchecked")
    public Set keySet();
  
```

```

//Връща множество от ключовете на текущата таблица

@SuppressWarnings("unchecked")
public Collection entryValues();
//Връща колекция от стойностите на текущата таблица

@SuppressWarnings("unchecked")
public Set entrySet();
//Връща множество от двойките от вида (ключ, стойност) на текущата таблица
}

```

Таблица не може да съдържа обекти-ключове и стойности, равни на **null**. Всеки метод с параметри key или value активира изключение **NullPointerException**, ако key=null или value=null.

Класът **Entry** представя ред на таблица – двойка от вида (ключ, стойност):

```

package Function;

public class Entry {
    // Data
    private Object key;
    private Object value;

    // Constructor
    public Entry(Object key) throws NullPointerException {
        if (key == null)
            throw new NullPointerException();
        this.key = key;
    }

    // Methods
    public Object getKey() {
        return key;
    }

    public Object getValue() {
        return value;
    }

    public Object setValue(Object newValue) throws NullPointerException {
        if (newValue == null)
            throw new NullPointerException();
        Object old = value;
        value = newValue;
        return old;
    }

    public boolean equals(Object obj) {
        Entry t = (Entry) obj;
        return key.equals(t.getKey()) && value.equals(t.getValue());
    }

    public String toString() {
        return "(" + key + "," + value + ")";
    }
}

```

Да се реализират:

1. Клас **AbstractTable**, съгласно спецификацията:

```

package Function;

import java.util.Collection;

```

```

import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import Interface.Function;

public abstract class AbstractTable implements Function {
    //Абстрактни методи
    public abstract Object get(Object key);
    public abstract Object put(Object key, Object value);
    public abstract Object remove(Object key);
    public abstract Iterator keys();
    public abstract Iterator values();

    //Реализация на методи на интерфейса Function
    public boolean containsKey(Object key) {...}
    public boolean containsValue(Object value) {...}
    public boolean isEmpty() {...}
    public boolean equals(Object obj) {...}
    public int size() {...}
    public Set keySet() {...}
    public Collection entryValues() {...}
    public Set entrySet() {...}

    //Предефиниране
    public String toString() {...}
}

```

Реализация:

```

package Function;

import java.util.Collection;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import Interface.Function;

public abstract class AbstractTable implements Function {
    // Abstract methods
    public abstract Object get(Object key);

    public abstract Object put(Object key, Object value);

    public abstract Object remove(Object key);

    @SuppressWarnings("unchecked")
    public abstract Iterator keys();

    @SuppressWarnings("unchecked")
    public abstract Iterator values();

    // Function methods implementation
    public boolean containsKey(Object key) {
        return keySet().contains(key);
    }

    public boolean containsValue(Object value) {
        throw new UnsupportedOperationException();
    }

    public boolean isEmpty() {
        return entrySet().isEmpty();
    }

    public boolean equals(Object obj) {

```

```

        throw new UnsupportedOperationException();
    }

    public int size() {
        return entrySet().size();
    }

    @SuppressWarnings("unchecked")
    public Set keySet() {
        Set result = new HashSet();
        Iterator it = keys();
        while (it.hasNext())
            result.add(it.next());
        return result;
    }

    @SuppressWarnings("unchecked")
    public Collection entryValues() {
        throw new UnsupportedOperationException();
    }

    @SuppressWarnings("unchecked")
    public Set entrySet() {
        Set result = new HashSet();
        Iterator it = keys();
        while (it.hasNext()) {
            Object key = it.next();
            Entry m = new Entry(key);
            m.setValue(get(key));
            result.add(m);
        }
        return result;
    }

    // Override
    @SuppressWarnings("unchecked")
    public String toString() {
        String result = "";
        Iterator it = keys();
        while (it.hasNext()) {
            Object key = it.next();
            Entry m = new Entry(key);
            m.setValue(get(key));
            result = result + m + "\n";
        }
        return result;
    }
}

```

2. Клас **ListTable**, съгласно спецификацията:

```

package Function;

import java.util.Iterator;
import java.util.LinkedList;
import Interface.Function;
import Iterator.TableKeysIterator;
import Iterator.TableValuesIterator;

public class ListTable extends AbstractTable implements Function {
    //Представяне на таблица
    private LinkedList list;

    //Конструктор
    public ListTable() { list = new LinkedList(); }

    ...
}

```

```
//Метод за достъп
public LinkedList getList() { return list; }

//Реализация на абстрактните методи на класа AbstractTable
//Наследяване на методите на интерфейса Function, реализирани в класа AbstractTable
}
```

Реализация:

```
package Function;

import java.util.Iterator;
import java.util.LinkedList;
import Interface.Function;
import Iterator.TableKeysIterator;
import Iterator.TableValuesIterator;

public class ListTable extends AbstractTable implements Function {
    // Data
    @SuppressWarnings("unchecked")
    private LinkedList list;

    // Constructors
    @SuppressWarnings("unchecked")
    public ListTable() {
        list = new LinkedList();
    }

    // Access method
    @SuppressWarnings("unchecked")
    public LinkedList getList() {
        return list;
    }

    // Abstract methods implementation
    @SuppressWarnings("unchecked")
    public Object get(Object key) {
        Iterator it = list.iterator();
        while (it.hasNext()) {
            Entry r = (Entry) (it.next());
            if (key.equals(r.getKey()))
                return r.getValue();
        }
        return null;
    }

    @SuppressWarnings("unchecked")
    public Object put(Object key, Object value) {
        Object result = null;
        Iterator it = list.iterator();
        while (it.hasNext()) {
            Entry r = (Entry) (it.next());
            if (key.equals(r.getKey())) {
                result = r.getValue();
                r.setValue(value);
            }
        }

        if (result == null) {
            Entry m = new Entry(key);
            m.setValue(value);
            list.addFirst(m);
        }

        return result;
    }
}
```

```

}

public Object remove(Object key) {
    Object value = get(key);
    if (value != null) {
        Entry m = new Entry(key);
        m.setValue(value);
        list.remove(m);
    }
    return value;
}

@SuppressWarnings("unchecked")
public Iterator keys() {
    return new TableKeysIterator(list.iterator());
}

@SuppressWarnings("unchecked")
public Iterator values() {
    return new TableValuesIterator(list.iterator());
}

// Function methods - inherited from class AbstractTable
}

```

3. Клас **HashTable**, съгласно спецификацията:

```

package Function;

import java.util.Hashtable;
import java.util.Iterator;
import Interface.Function;

public class HashTable extends AbstractTable implements Function {
    //Представяне на таблица
    private Hashtable hashtable;

    //Конструктор
    public HashTable() { hashtable = new Hashtable(); }

    //Реализация на абстрактните методи на класа AbstractTable
    //Наследяване на методите на интерфейса Function, реализирани в класа AbstractTable
}

```

Реализация:

```

package Function;

import java.util.Hashtable;
import java.util.Iterator;
import Interface.Function;

public class HashTable extends AbstractTable implements Function {
    // Data
    @SuppressWarnings("unchecked")
    private Hashtable hashtable;

    // Constructor
    @SuppressWarnings("unchecked")
    public HashTable() {
        hashtable = new Hashtable();
    }

    // Abstract methods implementation
    public Object get(Object key) {

```

```

        return hashtable.get(key);
    }

@SuppressWarnings("unchecked")
public Object put(Object key, Object value) {
    return hashtable.put(key, value);
}

public Object remove(Object key) {
    return hashtable.remove(key);
}

@SuppressWarnings("unchecked")
public Iterator keys() {
    return hashtable.keySet().iterator();
}

@SuppressWarnings("unchecked")
public Iterator values() {
    throw new UnsupportedOperationException();
}

// Function methods - inherited from class AbstractTable
}

```

При реализацията се използва родовата хеш-таблица **java.util.Hashtable<K,V>** - виж <http://java.sun.com/javase/6/docs/api/>.

4. Клас TreeTable, съгласно спецификацията:

```

package Function;

import java.util.Iterator;
import Interface.Function;
import Iterator.TableKeysIterator;
import Iterator.TableValuesIterator;
import Tree.BinaryTree;

public class TreeTable extends AbstractTable implements Function {
    //Представяне на таблица
    private BinaryTree tree;

    //Конструктор
    public TreeTable(BinaryTree arg) { tree = arg; }

    //Реализация на абстрактните методи на класа AbstractTable
    //Наследяване на методите на интерфейса Function, реализирани в класа AbstractTable
}

```

Реализация:

```

package Function;

import java.util.Iterator;
import Interface.Function;
import Iterator.TableKeysIterator;
import Iterator.TableValuesIterator;
import Tree.BinaryTree;

public class TreeTable extends AbstractTable implements Function {
    // Data
    private BinaryTree tree;

    // Constructor

```

```

public TreeTable(BinaryTree arg) {
    tree = arg;
}

// Access method
public BinaryTree getTree() {
    return tree;
}

// Abstract methods implementation
@SuppressWarnings("unchecked")
public Object get(Object key) {
    Object result = null;
    Iterator it = tree.iterator();
    while (it.hasNext()) {
        Entry temp = (Entry) it.next();
        if (temp.getKey().equals(key))
            result = temp.getValue();
    }
    return result;
}

@SuppressWarnings("unchecked")
public Object put(Object key, Object value) {
    Object result = null;
    Iterator it = tree.iterator();
    while (it.hasNext()) {
        Entry r = (Entry) (it.next());
        if (key.equals(r.getKey())) {
            result = r.getValue();
            r.setValue(value);
        }
    }

    if (result == null) {
        Entry m = new Entry(key);
        m.setValue(value);
        tree.add(m);
    }

    return result;
}

public Object remove(Object key) {
    Entry m = (Entry) (tree.remove(new Entry(key)));
    if (m == null)
        return null;
    return m.getValue();
}

@SuppressWarnings("unchecked")
public Iterator keys() {
    return new TableKeysIterator(tree.iterator());
}

@SuppressWarnings("unchecked")
public Iterator values() {
    return new TableValuesIterator(tree.iterator());
}

// Function methods - inherited from class AbstractTable
}

```

5. Клас TableValuesIterator за обхождане на стойностите на таблица

Реализация:

```
package Iterator;

import java.util.Iterator;
import Function.Entry;

@SuppressWarnings("unchecked")
public class TableValuesIterator implements Iterator {
    // Data
    Iterator it, itOld;

    // Constructor
    public TableValuesIterator(Iterator arg) {
        it = itOld = arg;
    }

    // Iterator methods
    public boolean hasNext() {
        return it.hasNext();
    }

    public Object next() {
        return ((Entry) (it.next())).getValue();
    }

    public void remove() {
        it.remove();
    }

    public void reset() {
        it = itOld;
    }
}
```

6. Клас TableKeysIterator за обхождане на ключовете на таблица

Реализация:

```
package Iterator;

import java.util.Iterator;
import Function.Entry;

@SuppressWarnings("unchecked")
public class TableKeysIterator implements Iterator {
    // Data
    Iterator it, itOld;

    // Constructor
    public TableKeysIterator(Iterator arg) {
        it = itOld = arg;
    }

    // Iterator methods
    public boolean hasNext() {
        return it.hasNext();
    }

    public Object next() {
        return ((Entry) (it.next())).getKey();
    }

    public void remove() {
        it.remove();
    }
}
```

```

    }

    public void reset() {
        it = itOld;
    }
}

```

При реализацията се използва таблицата за основните интерфейси и тяхната реализация - виж <http://java.sun.com/docs/books/tutorial/>:

General-purpose Implementations					
Interfaces	Implementations				
	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap		LinkedHashMap

Пример: Нека table е обект на клас, реализиращ интерфейса **Function**. Следващата таблица съдържа операции за включване в table, които се изпълняват в реда от таблицата. Хешкодът w.hashCode() на името $w = c_0c_1\dots c_{n-1}$ (обект на класа **String**), се определя по следния начин:

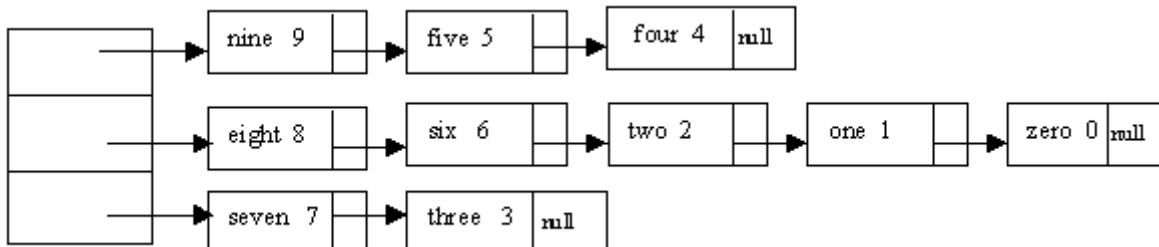
$$c_0 * 31^{(n-1)} + c_1 * 31^{(n-2)} + \dots + c_{n-1}$$

Име на цифра на английски език	Цифра	Операция	Сума от номерата на буквите на името	Сума % 3	Хешкод на името	Хешкод % 3
Zero	0	table.put("zero",new Integer(0))	448	1	3735208	1
One	1	table.put("one",new Integer(1))	322	1	110182	1
Two	2	table.put("two",new Integer(2))	346	1	115276	1
Three	3	table.put("three",new Integer(3))	536	2	110339486	2
Four	4	table.put("four",new Integer(4))	444	0	3149094	0
Five	5	table.put("five",new Integer(5))	426	0	3143346	0
Six	6	table.put("six",new Integer(6))	340	1	113890	1
Seven	7	table.put("seven",new Integer(7))	545	2	109330445	2
Eight	8	table.put("eight",new Integer(8))	529	1	96505999	1
Nine	9	table.put("nine",new Integer(9))	426	0	3381426	0

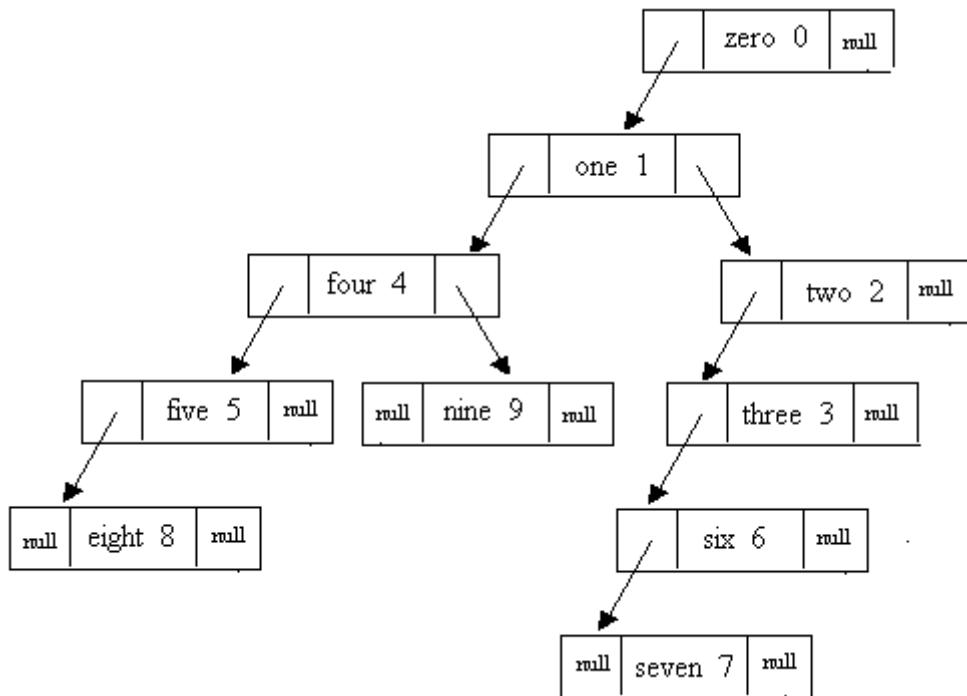
1. table = new ListTable() – представяне чрез линеен едносвързан списък



2. table = new HashTable() – представяне чрез масив от три линейни едносвързани списъка



3. `table = new TreeTable(new BinarySearchTree(new EntryComparator()))` – представяне чрез двоично дърво на търсене



Тестване: Програмата **TableTester**:

1. Чете низове от вида: дума@цяло_число от файла `testTableData.txt` и създава таблица от двойки (дума, цяло_число)

Function table = new ListTable()

Извежда елементите на таблицата.

2. Чете низове от вида: дума@цяло_число от файла `testTableData.txt` и създава таблица от двойки (дума, цяло_число)

Function table = new HashTable()

Извежда елементите на таблицата.

3. Чете низове от вида: дума@цяло_число от файла `testTableData.txt` и създава таблица от двойки (дума, цяло_число)

BinaryTree tree = new BinarySearchTree(new EntryComparator());
Function table = new TreeTable(tree);

Обхожда дървото по четирите начина и записва резултата във файла `testTreeTableResult.txt`.

4. Обхожда дървото от т.3, като използва итератора

Iterator it = new TableValuesIterator(tree.iterator());

и извежда резултата.

```

package Test;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import Comparator.EntryComparator;
import Function.HashTable;
import Function.ListTable;
import Function.TreeTable;
import Interface.Function;
import Iterator.TableValuesIterator;
import Tree.BinarySearchTree;
import Tree.BinaryTree;
  
```

```

public class TableTester {
    public void createTable(Function table, String source) throws IOException {
        BufferedReader in = new BufferedReader(new FileReader(source));

        String line = in.readLine();
        while (!line.equals("end")) {
            int index = line.indexOf('@');
            String word = line.substring(0, index);
            String digit = line.substring(index + 1);
            table.put(word, Integer.valueOf(digit));
            line = in.readLine();
        }

        in.close();
    }

    public void testListTable() {
        try {
            String s = File.separator;
            String userDir = System.getProperty("user.dir");
            System.out.println("User directory: " + userDir);
            String dir = userDir + s + "src" + s + "Test";

            String source = dir + s + "testTableData.txt";
            System.out.println("Read text from the file " + source);
            Function table = new ListTable();
            createTable(table, source);
            System.out.println("List table Digits in English:");
            System.out.println(table);

        } catch (IOException e) {
        }
    }

    public void testHashTable() {
        try {
            String s = File.separator;
            String userDir = System.getProperty("user.dir");
            System.out.println("User directory: " + userDir);
            String dir = userDir + s + "src" + s + "Test";

            String source = dir + s + "testTableData.txt";
            System.out.println("Read text from the file " + source);

            Function table = new HashTable();
            createTable(table, source);
            System.out.println("Hash table Digits in English:" + "\n"
                + table);

        } catch (IOException e) {
        }
    }

    @SuppressWarnings("unchecked")
    public void testTreeTable() {
        try {
            String s = File.separator;
            String userDir = System.getProperty("user.dir");
            System.out.println("User directory: " + userDir);
            String dir = userDir + s + "src" + s + "Test";

            String source = dir + s + "testTableData.txt";
            System.out.println("Read text from the file " + source);

            String target = dir + s + "testTreeTableResult.txt";
            System.out.println("Test and write results to the file " + target);
        }
    }
}

```

```

BinaryTree tree = new BinarySearchTree(new EntryComparator());
Function table = new TreeTable(tree);
createTable(table, source);
new BinaryTreeTester().test(tree, target);

System.out.println(" Tree table Digits traversal:");
Iterator it = new TableValuesIterator(tree.iterator());
while (it.hasNext())
    System.out.println(it.next());

} catch (IOException e) {
}
}

public static void main(String[] args) {
    TableTester tester = new TableTester();
    tester.testListTable();
    tester.testHashTable();
    tester.testTreeTable();
}
}

```

Резултати от извеждането:

```

User directory: D:\UserFolder\JavaProjects\SDP
Read text from the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTableData.txt
    List table Digits in English:
(nine,9)
(eight,8)
(seven,7)
(six,6)
(five,5)
(four,4)
(three,3)
(two,2)
(one,1)
(zero,0)

User directory: D:\UserFolder\JavaProjects\SDP
Read text from the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTableData.txt
    Hash table Digits in English:
(three,3)
(six,6)
(seven,7)
(nine,9)
(one,1)
(zero,0)
(five,5)
(four,4)
(eight,8)
(two,2)

User directory: D:\UserFolder\JavaProjects\SDP
Read text from the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTableData.txt
Test and write results to the file
D:\UserFolder\JavaProjects\SDP\src\Test\testTreeTableResult.txt
    Tree table Digits traversal:
8
5
4
9
1

```

7
6
3
2
0

Файл testTableData.txt

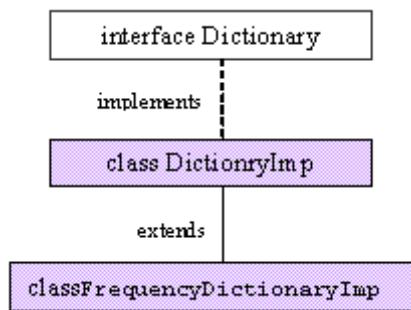
```
zero@0
one@1
two@2
three@3
four@4
five@5
six@6
seven@7
eight@8
nine@9
end
```

Файл testTreeTableResult.txt

```
Testing...
Height of the tree: 5
    Tree breadth-first traversal:
        Level #: 0
        (zero,0)
        Level #: 1
        (one,1)
        Level #: 2
        (four,4)
        (two,2)
        Level #: 3
        (five,5)
        (nine,9)
        (three,3)
        Level #: 4
        (eight,8)
        (six,6)
        Level #: 5
        (seven,7)
        Tree inorder traversal:
        (eight,8)
        (five,5)
        (four,4)
        (nine,9)
        (one,1)
        (seven,7)
        (six,6)
        (three,3)
        (two,2)
        (zero,0)
        Tree preorder traversal:
        (zero,0)
        (one,1)
        (four,4)
        (five,5)
        (eight,8)
        (nine,9)
        (two,2)
        (three,3)
        (six,6)
        (seven,7)
        Tree postorder traversal:
        (eight,8)
        (five,5)
        (nine,9)
        (four,4)
        (seven,7)
        (six,6)
        (three,3)
        (two,2)
        (one,1)
        (zero,0)
Done!
```



Задача: Дадена е йерархията от интерфейси и класове:



Интерфейсът **Dictionary** представя операции на *речник* - таблица с ключове от тип **String** и стойности от тип **Object**:

```

package Interface;

import java.util.Collection;
import java.util.Iterator;
import java.util.Set;

public interface Dictionary {
    public boolean containsKey(String key);
    //Проверява дали в текущата таблица се съдържа ключът key

    public boolean containsValue(Object value);
    //Проверява дали в текущата таблица се съдържа стойността value

    public Object get(String key);
    //Връща стойността на функцията за ключа key от текущата таблица

    public Object put(String key, Object value);
    //Ако има ред с ключ key в текущата таблица, съответната стойност се променя
    //на value. В противен случай се включва нов ред (key,value) в текущата
    //таблица

    public Object remove(String key);
    //Изключва ред с ключ key от текущата таблица и връща съответната стойност

    public boolean isEmpty();
    //Проверява дали текущата таблица е празна

    public boolean equals(Object obj);
    //Проверява дали текущата таблица и таблицата obj представлят една и съща
    //функция

    public int size();
    //Връща броя на редовете на текущата таблица

    @SuppressWarnings("unchecked")
    public Iterator keys();
    //Обхожда ключовете на текущата таблица

    @SuppressWarnings("unchecked")
    public Iterator values();
    //Обхожда стойностите на текущата таблица

    @SuppressWarnings("unchecked")
    public Set keySet();
    //Връща множество от ключовете на текущата таблица
  
```

```

@SuppressWarnings("unchecked")
public Collection entryValues();
//Връща колекция от стойностите на текущата таблица

@SuppressWarnings("unchecked")
public Set entrySet();
//Връща множество от двойките от вида (ключ, стойност) на текущата таблица
}

```

1. Да се реализира клас **DictionaryImp** за представяне на речник, съгласно спецификацията:

```

package Dictionary;

import java.util.Iterator;
import Comparator.EntryComparator;
import Function.Entry;
import Interface.Dictionary;
import Interface.Function;
import Iterator.TableKeysIterator;
import List.SortedLinkedList;

public class DictionaryImp implements Dictionary {
    //Представяне на речник
    private Function table;

    //Конструктор
    public DictionaryImp(Function table) { this.table = table; }

    //Реализация на методите на интерфейса Dictionary
}

```

Реализация:

```

package Dictionary;

import java.util.Comparator;
import java.util.Iterator;
import Comparator.EntryComparator;
import Function.Entry;
import Interface.Dictionary;
import Interface.Function;
import Iterator.TableKeysIterator;
import List.SortedLinkedList;

public class DictionaryImp implements Dictionary {
    // Data
    private Function table;

    // Constructor
    public DictionaryImp(Function table) {
        this.table = table;
    }

    // Dictionary methods implementation
    public boolean containsKey(String key) {
        return table.containsKey(key);
    }

    public boolean containsValue(Object value) {
        throw new UnsupportedOperationException();
    }
}

```

```

public Object get(String key) {
    return table.get(key);
}

public Object put(String key, Object value) {
    return table.put(key, value);
}

public Object remove(String key) {
    return table.remove(key);
}

public boolean isEmpty() {
    return table.isEmpty();
}

public boolean equals(Object obj) {
    throw new UnsupportedOperationException();
}

public int size() {
    return table.size();
}

@SuppressWarnings("unchecked")
public Iterator keys() {
    Comparator c = new EntryComparator();
    SortedLinkedList temp = new SortedLinkedList();
    // java.util.Set temp = new java.util.TreeSet(c);
    Iterator it = table.keys();
    while (it.hasNext()) {
        String key = (String) it.next();
        Entry m = new Entry(key);
        m.setValue(get(key));
        temp.addInSortedList(c, m);
        // temp.add(m);
    }
    return new TableKeysIterator(temp.iterator());
}

@SuppressWarnings("unchecked")
public Iterator values() {
    return table.values();
}

@SuppressWarnings("unchecked")
public String toString() {
    String result = "";
    Iterator it = keys();
    while (it.hasNext()) {
        String key = (String) it.next();
        Entry m = new Entry(key);
        m.setValue(get(key));
        result = result + m + "\n";
    }
    return result;
}
}

```

2. Да се илюстрира използването на класа **DictionaryImp** за създаване и извеждане на следните речници:

2.1. *Речник D(L₁,L₂)={ (a₁,b₁),(a₂,b₂),..., (a_n,b_n) } за превод от език L₁ на език L₂, където:*

- a₁,a₂,...,a_n са думи на езика L₁
- b₁,b₂,...,b_n са съответните им думи на езика L₂

2.2. *Честотен речник D(T)={(ω₁,n₁/p),(ω₂,n₂/p),..., (ω_k,n_k/p)} на текст T, където:*

- $\omega_1, \dots, \omega_k$ са различните думи в текста
- n_1 -брой срещания на думата ω_1 в текста, n_2 -брой срещания на думата ω_2 в текста, ..., n_k -брой срещания на думата ω_k в текста
- p е общият брой думи в текста

Реализация на честотен речник:

```
package Dictionary;

import java.util.Iterator;
import Interface.Dictionary;
import Interface.Function;

public class FrequencyDictionaryImp extends DictionaryImp implements Dictionary
{
    // Data
    private boolean created;
    private int count;

    // Constructor
    public FrequencyDictionaryImp(Function table) {
        super(table);
        created = false;
        count = 0;
    }

    // Public method
    @SuppressWarnings("unchecked")
    public FrequencyDictionaryImp create() {
        Iterator it = super.keys();
        while (it.hasNext()) {
            String word = (String) (it.next());
            Object value = get(word);
            double frequency = ((Double) value).doubleValue() / count;
            super.put(word, (new Double(frequency)));
        }

        created = true;
        return this;
    }

    // Dictionary methods - inherited from DictionaryImp

    // Override method
    public Object put(String word, Object value) {
        if (created)
            throw new UnsupportedOperationException("Dictionary is created!");
        Double valueNew = ((Double) value);
        count += valueNew;
        Object valueOld = super.get(word);
        if (valueOld != null)
            valueNew = new Double(((Double) valueOld).doubleValue()
                + ((Double) valueNew).doubleValue());
        super.put(word, valueNew);
        return valueNew;
    }
}
```

Тестване: Програмата **DictionaryTester**:

1. Чете низове от вида: дума1@дума2 от файла testDictData.txt и създава речник от двойки (дума1,дума2)

Dictionary dict1 = new DictionaryImp(new ListTable())

Извежда речника, сортиран в азбучен ред на първите думи в него.

2. Чете текст от файла testFreqDictData.txt и създава честотен речник на текста, състоящ се от двойки (дума,честота_на_срещане)

FrequencyDictionaryImp dict2 = new FrequencyDictionaryImp(new ListTable())

Извежда речника, сортиран в азбучен ред на думите в него.

```

package Test;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import Dictionary.DictionaryImp;
import Dictionary.FrequencyDictionaryImp;
import Function.ListTable;
import Interface.Dictionary;
import Interface.Function;

public class DictionaryTester {
    public void createDictionary(Dictionary dict, String source)
        throws IOException {
        BufferedReader in = new BufferedReader(new FileReader(source));

        String line = in.readLine();
        while (!line.equals("end")) {
            int index = line.indexOf('@');
            String wordEng = line.substring(0, index);
            String wordBul = line.substring(index + 1);
            dict.put(wordEng, wordBul);
            line = in.readLine();
        }
    }

    public void createFrequencyDictionary(FrequencyDictionaryImp dict,
        String source) throws IOException {
        BufferedReader in = new BufferedReader(new FileReader(source));

        String word = "";
        int c;
        while ((c = in.read()) != -1) {
            for (; (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'); c = in
                .read())
                word = word + (char) c;

            if (word.compareTo("") != 0) {
                dict.put(word, new Double(1));
                word = "";
            }

            if (c == -1)
                break;
        }
        dict.create();
    }

    public static void main(String[] args) {
        try {
            String s = File.separator;
            String userDir = System.getProperty("user.dir");
            System.out.println("User directory: " + userDir);
            String dir = userDir + s + "src" + s + "Test";

            String source = dir + s + "testDictData.txt";
            System.out.println("Read text from the file " + source);
            Function table = new ListTable();
            // table = new TreeTable(new BinarySearchTree(new

```

```
// EntryComparator())));
// table = new HashTable();
Dictionary dict1 = new DictionaryImp(table);
DictionaryTester tester = new DictionaryTester();
tester.createDictionary(dict1, source);
System.out.println("      English-Bulgarian Dictionary:" + "\n"
+ dict1);

source = dir + s + "testFreqDictData.txt";
System.out.println("Read text from the file " + source);

table = new ListTable();
// table = new TreeTable(new BinarySearchTree(new
// EntryComparator()));
// table = new HashTable();
FrequencyDictionaryImp dict2 = new FrequencyDictionaryImp(table);
tester.createFrequencyDictionary(dict2, source);
System.out.println("      Frequency Dictionary:" + "\n" + dict2);

} catch (IOException e) {
}
}
```

Резултати от тестването:

```
User directory: D:\UserFolder\JavaProjects\SDP
Read text from the file D:\UserFolder\JavaProjects\SDP\src\Test\testDictData.txt
    English-Bulgarian Dictionary:
(black,4eren)
(blue,sin)
(green,zelen)
(purple,morav)
(red,4erven)
(white,bial)
(yellow,jalt)

Read text from the file
D:\UserFolder\JavaProjects\SDP\src\Test\testFreqDictData.txt
    Frequency Dictionary:
(Constructor,0.05)
(Data,0.05)
(Object,0.1)
(TreeNode,0.15)
(class,0.05)
(d,0.1)
(data,0.1)
(left,0.1)
(null,0.05)
(protected,0.1)
(public,0.05)
(right,0.1)
```

Файл testDictData.txt

```
black@4eren  
red@4erven  
white@bial  
purple@morav  
green@zelen  
blue@sin  
yellow@jalt  
end
```

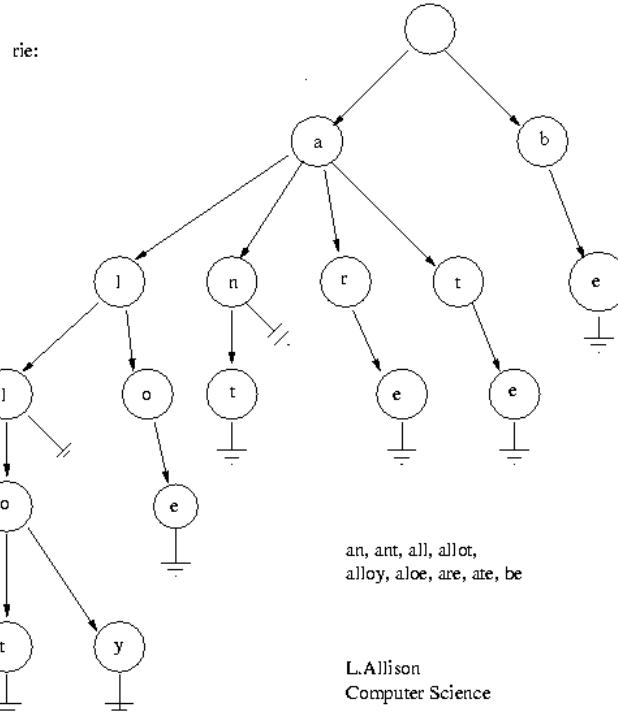
Файл testFreqDictData.txt

```
class TreeNode {  
    //Data  
    protected Object data;  
    protected TreeNode left, right;  
  
    //Constructor  
    public TreeNode(Object d) {  
        data=d; left=right=null;  
    }  
}
```



Префиксно дърво

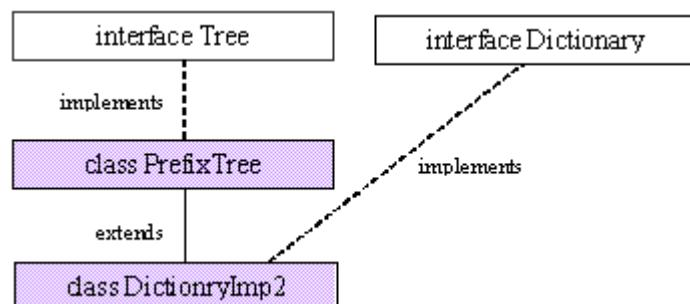
За представяне на речник може да се използва *префиксно дърво* (*trie, prefix tree*) – виж <http://en.wikipedia.org/wiki/Trie>. За разлика от двоичното дърво на търсене, връх на префиксно дърво не съдържа ключ, а символ, като с всеки връх се асоциира низ. Коренът се асоциира с празния низ. Низовете, асоциирани с наследниците на връх имат общ префикс – низа, асоцииран с върха. На следващата диаграма е представено префиксно дърво, съдържащо ключовете an, ant, all, allot, alloy, aloe, are, ate и be. Всеки връх, с който се асоциира ключ, е маркиран.



Следващата таблица съдържа върхове на дървото и асоциираните с тях низове:

Връх	Низ
Корен на дървото	""
a	a
l	al
n	an
t	ant (ключ)
b	b
e	be (ключ)
y	alloy (ключ)

Задача: Дадена е йерархията от интерфейси и класове:



Да се реализират:

1. Клас **PrefixTree** – префиксно дърво от ключове (обекти на класа **String**), съставени от малките латински букви, съгласно спецификацията:

```
package PrefixTree;

import java.util.Iterator;
import Interface.Tree;
import Iterator.PrefixTreeBreadthFirstIterator;
import Iterator.PrefixTreePreorderIterator;

public class PrefixTree implements Tree {
    //Данни
    public char content;
    public Object marker; //The marker denotes the end of a key
    public PrefixTree[] child;

    //Конструтори
    public PrefixTree() {
        content = ' '; //Root contains blank - represents empty string
        marker = null;
        child = new PrefixTree[26]; //alphabet = "abcdefghijklmnopqrstuvwxyz"
    }

    public PrefixTree(int i) {
        content = (char)('a' + i);
        marker = null;
        child = new PrefixTree[26]; //alphabet = "abcdefghijklmnopqrstuvwxyz"
    }

    //Методи за достъп
    public char character() { return content; }
    public boolean endOfKey() { return marker != null; }
    public Object data() { return marker; }
    public PrefixTree[] child() { return child; }

    //Protected методи
    protected PrefixTree insert(String s) {
        PrefixTree current = this;
        PrefixTree result = null;

        if(s.length() == 0) //For empty String
            current.marker = new Boolean(true);

        System.out.println(" Inserted characters:");
        for(int i = 0; i < s.length(); i++) {
            if(current.child[s.charAt(i)-'a'] != null) {
                current = current.child[s.charAt(i)-'a'];
                System.out.print(current.content + " ");
            }
            else {
                result = new PrefixTree((int)(s.charAt(i)-'a'));
                current.child[s.charAt(i)-'a'] = result;
                current = current.child[s.charAt(i)-'a'];
                System.out.print(current.content + " ");
            }
            if(i == s.length()-1) {
                current.marker = new Boolean(true);
                result = current;
            }
        }
    }
}
```

```

        System.out.println("\nFinished inserting the word: " + s + "\n");
        return result;
    }

protected PrefixTree search(String s) {
    PrefixTree current = this;
    System.out.println("\n    Searching for string: " + s);

    while(current != null) {
        for(int i = 0; i < s.length(); i++) {
            if(current.child[s.charAt(i)-'a'] == null) {
                System.out.println("Cannot find string: " + s);
                return null;
            }
            else {
                current = current.child[s.charAt(i)-'a'];
                System.out.println("Found character: " + current.content);
            }
        }

        if (current.marker != null) {
            System.out.println("Found string: " + s);
            return current;
        }
        else {
            System.out.println("Cannot find string: " + s + "(only present as a substring)");
            return null;
        }
    }

    return null;
}

```

//Реализация на методите на интерфейса Tree

```

//Итератори
public Iterator treeBreadthFirstIterator() { return new PrefixTreeBreadthFirstIterator(this); }
public Iterator treePreorderIterator() { return new PrefixTreePreorderIterator(this); }

//Други методи
public String characters() {
    String result = "";
    Iterator<Character> it = treeBreadthFirstIterator();
    //...
    return result;
}

```

Реализация:

```

package PrefixTree;

import java.util.Iterator;
import Interface.Tree;
import Iterator.PrefixTreeBreadthFirstIterator;
import Iterator.PrefixTreePreorderIterator;

public class PrefixTree implements Tree {
    // Data
    public char content;
    public Object marker; // The marker denotes the end of a key
    public PrefixTree[] child;

    // Constructors
    public PrefixTree() {

```

```

content = ' '; // Root contains blank - represents empty string
marker = null;
child = new PrefixTree[26]; // alphabet = "abcdefghijklmnopqrstuvwxyz"
}

public PrefixTree(int i) {
    content = (char) ('a' + i);
    marker = null;
    child = new PrefixTree[26]; // alphabet = "abcdefghijklmnopqrstuvwxyz"
}

// Access methods
public char character() {
    return content;
}

public boolean endOfKey() {
    return marker != null;
}

public Object data() {
    return marker;
}

public PrefixTree[] child() {
    return child;
}

// Protected methods
protected PrefixTree insert(String s) {
    PrefixTree current = this;
    PrefixTree result = null;

    if (s.length() == 0) // For empty String
        current.marker = new Boolean(true);

    System.out.println("      Inserted characters:");
    for (int i = 0; i < s.length(); i++) {
        if (current.child[s.charAt(i) - 'a'] != null) {
            current = current.child[s.charAt(i) - 'a'];
            System.out.print(current.content + "    ");
        }
        else {
            result = new PrefixTree((int) (s.charAt(i) - 'a'));
            current.child[s.charAt(i) - 'a'] = result;
            current = current.child[s.charAt(i) - 'a'];
            System.out.print(current.content + "    ");
        }
        if (i == s.length() - 1) {
            current.marker = new Boolean(true);
            result = current;
        }
    }

    System.out.println("\nFinished inserting the word: " + s + "\n");
    return result;
}

protected PrefixTree search(String s) {
    PrefixTree current = this;
    System.out.println("\n      Searching for string: " + s);

    while (current != null) {
        for (int i = 0; i < s.length(); i++) {

```

```

    if (current.child[s.charAt(i) - 'a'] == null) {
        System.out.println("Cannot find string: " + s);
        return null;
    } else {
        current = current.child[s.charAt(i) - 'a'];
        System.out.println("Found character: " + current.content);
    }
}

if (current.marker != null) {
    System.out.println("Found string: " + s);
    return current;
} else {
    System.out.println("Cannot find string: " + s
        + "(only present as a substring)");
    return null;
}
}

return null;
}

// Tree methods implementation
public void add(Object value) {
    insert((String) value);
}

public boolean contains(Object value) {
    return search((String) value) != null;
}

public boolean isEmpty() {
    return size() == 0;
}

public int height() {
    PrefixTree[] child = child();
    int max = -1;
    for (int i = 0; i < child.length; i++) {
        PrefixTree t = child[i];
        if (t != null) {
            int h = t.height();
            if (max < h)
                max = h;
        }
    }
    return max + 1;
}

public int numberofLiefs() {
    throw new UnsupportedOperationException();
}

public Object clone() {
    throw new UnsupportedOperationException();
}

public Object remove(Object value) {
    throw new UnsupportedOperationException();
}

@SuppressWarnings("unchecked")
public int size() {
    int result = 0;
    Iterator<String> it = iterator();
    while (it.hasNext()) {

```

```

        it.next();
        result++;
    }
    return result;
}

public void clear() {
    throw new UnsupportedOperationException();
}

@SuppressWarnings("unchecked")
public Iterator iterator() {
    return treePreorderIterator();
}

@SuppressWarnings("unchecked")
public Iterator treeInorderIterator() {
    throw new UnsupportedOperationException();
}

@SuppressWarnings("unchecked")
public Iterator treePreorderIterator() {
    return new PrefixTreePreorderIterator(this);
}

@SuppressWarnings("unchecked")
public Iterator treeBreadthFirstIterator() {
    return new PrefixTreeBreadthFirstIterator(this);
}

@SuppressWarnings("unchecked")
public Iterator treePostorderIterator() {
    throw new UnsupportedOperationException();
}

@SuppressWarnings("unchecked")
public Iterator subtreesIterator() {
    throw new UnsupportedOperationException();
}

@SuppressWarnings("unchecked")
public String toString() {
    String result = "";
    java.util.Iterator<String> it = iterator();
    while (it.hasNext())
        result = result + it.next() + "\n";
    return result;
}

// Other methods
@SuppressWarnings("unchecked")
public String characters() {
    String result = "";
    Iterator<Character> it = treeBreadthFirstIterator();
    PrefixTreeBreadthFirstIterator itw = (PrefixTreeBreadthFirstIterator) it;
    for (int i = 0; i < itw.numberOfLevels(); i++) {
        Iterator<Character> currentLevelIt = itw.getData(i).iterator();
        result = result + "\nLevel #: " + i + "\n";
        while (currentLevelIt.hasNext())
            result = result + currentLevelIt.next() + " ";
    }
    return result;
}
}

```

Класът **PrefixTreePreorderIterator** представя итератор за обхождане на ключовете на префиксно дърво в азбучен ред:

```

package Iterator;

import java.util.Iterator;
import java.util.LinkedList;

import PrefixTree.PrefixTree;

public class PrefixTreePreorderIterator implements Iterator<String> {
    // Data
    LinkedList<String> list;
    Iterator<String> it;

    // Constructor
    public PrefixTreePreorderIterator(PrefixTree tree) {
        list = new LinkedList<String>();
        preorder(tree, "");
        reset();
    }

    // Private method
    private void preorder(PrefixTree tree, String s) {
        // Root visit
        s = s + tree.character();
        if (tree.endOfKey()) {
            if (s.length() == 0)
                list.addLast("");
            else
                list.addLast(s.substring(1));
        }
        // Subtrees visit
        PrefixTree[] child = tree.child();
        for (int i = 0; i < child.length; i++) {
            tree = child[i];
            if (tree != null)
                preorder(tree, s);
        }
    }

    // Iterator methods
    public boolean hasNext() {
        return it.hasNext();
    }

    public String next() {
        return it.next();
    }

    public void reset() {
        it = list.iterator();
    }

    public void remove() {
        it.remove();
    }
}

```

Класът **PrefixTreeBreadthFirstIterator** представя итератор за обхождане на символите на префиксно дърво по нива

```
package Iterator;
```

```

import java.util.Iterator;
import java.util.LinkedList;
import java.util.NoSuchElementException;
import java.util.Vector;
import PrefixTree.PrefixTree;

public class PrefixTreeBreadthFirstIterator implements Iterator<Character> {
    // Data
    Vector<LinkedList<Character>> levels; // Vector of levels
    Iterator<Character> it;
    int level;

    // Constructor
    public PrefixTreeBreadthFirstIterator(PrefixTree tree) {
        levels = new Vector<LinkedList<Character>>();
        breadthFirstTraversal(tree);
        reset();
    }

    // Private method
    private void breadthFirstTraversal(PrefixTree tree) {
        LinkedList<PrefixTree> currentLevel = new LinkedList<PrefixTree>();
        if (!tree.isEmpty())
            currentLevel.addLast(tree);
        while (!currentLevel.isEmpty()) {
            LinkedList<Character> data = new LinkedList<Character>();
            LinkedList<PrefixTree> nextLevel = new LinkedList<PrefixTree>();
            Iterator<PrefixTree> it = currentLevel.iterator();
            while (it.hasNext()) {
                PrefixTree t = it.next();
                data.addLast(t.character());
                PrefixTree[] child = t.child();
                for (int i = 0; i < child.length; i++)
                    if (t.child[i] != null)
                        nextLevel.addLast(t.child[i]);
            }
            levels.add(data);
            level++;
            currentLevel = nextLevel;
        }
    }

    // Methods
    public int numberOfLevels() {
        return levels.size();
    }

    public LinkedList<Character> getData(int level) {
        return levels.get(level);
    }

    // Iterator methods
    public boolean hasNext() {
        if (it.hasNext())
            return true;
        level++;
        if (level < levels.size()) {
            it = levels.get(level).iterator();
            return true;
        }
        return false;
    }

    public Character next() {
        if (!hasNext())
            throw new NoSuchElementException();
    }
}

```

```

        return it.next();
    }

public void remove() {
    throw new UnsupportedOperationException();
}

public void reset() {
    level = 0;
    it = levels.get(0).iterator();
}
}
}

```

Тестване: Програмата **PrefixTreeTester** чете думите an, ant, all, allot, alloy, aloe, are, ate и be от текстов файл testPrefixTreeData.txt и създава префиксното дърво, представено в началото. Обхожда дървото с итераторите от т.2 и т.3 и записва резултата във файла testPrefixTreeResult.txt.

```

package Test;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import PrefixTree.PrefixTree;

public class PrefixTreeTester {
    public void createTree(PrefixTree tree, String source) throws IOException {
        BufferedReader in = new BufferedReader(new FileReader(source));

        String line = in.readLine();
        while (!line.equals("end")) {
            tree.add(line);
            line = in.readLine();
        }

        in.close();
    }

    public void test(PrefixTree tree, String target) throws IOException {
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(
            target)));
        out.println("    Height of the tree: " + tree.height());
        out.println("    Prefix Tree-breadth-first traversal:");
        out.println(tree.characters());

        out.println("    Size of the tree: " + tree.size());
        out.println("    Prefix Tree-preorder traversal:");
        out.println(tree);

        out.close();
    }

    public static void main(String[] args) {
        try {
            String s = File.separator;
            String userDir = System.getProperty("user.dir");
            System.out.println("User directory: " + userDir);
            String dir = userDir + s + "src" + s + "Test";

            String source = dir + s + "testPrefixTreeData.txt";

```

```
System.out.println("Read text from the file " + source);

PrefixTree tree = new PrefixTree();
PrefixTreeTester tester = new PrefixTreeTester();
tester.createTree(tree, source);

String target = dir + s + "tesPrefixTreeResult.txt";
System.out.println("Test and write results to the file " + target);
tester.test(tree, target);

tree.contains("all");
tree.contains("alloy");
tree.contains("al");
tree.contains("ask");
tree.contains("");

} catch (IOException e) {
}
}
```

Резултати от тестването:

```
User directory: D:\UserFolder\JavaProjects\SDP
Read text from the file
D:\UserFolder\JavaProjects\SDP\src\Test\testPrefixTreeData.txt
    Inserted characters:
a   n
Finished inserting the word: an

    Inserted characters:
a   n   t
Finished inserting the word: ant

    Inserted characters:
a   l   l
Finished inserting the word: all

    Inserted characters:
a   l   l   o   t
Finished inserting the word: allot

    Inserted characters:
a   l   l   o   y
Finished inserting the word: alloy

    Inserted characters:
a   l   o   e
Finished inserting the word: aloe

    Inserted characters:
a   r   e
Finished inserting the word: are

    Inserted characters:
a   t   e
Finished inserting the word: ate

    Inserted characters:
b   e
Finished inserting the word: be

Test and write results to the file
D:\UserFolder\JavaProjects\SDP\src\Test\tesPrefixTreeResult.txt

    Searching for string: all
```

```

Found character: a
Found character: l
Found character: l
Found string: all

    Searching for string: alloy
Found character: a
Found character: l
Found character: l
Found character: o
Found character: y
Found string: alloy

    Searching for string: al
Found character: a
Found character: l
Cannot find string: al(only present as a substring)

    Searching for string: ask
Found character: a
Cannot find string: ask

    Searching for string:
Cannot find string: (only present as a substring)

```

Файл testPrefixTreeData.txt	Файл testPrefixTreeResult.txt
an ant all allot alloy aloe are ate be end	Height of the tree: 5 Prefix Tree-breadth-first traversal: Level #: 0 Level #: 1 a b Level #: 2 l n r t e Level #: 3 l o t e e Level #: 4 o e Level #: 5 t y Size of the tree: 9 Prefix Tree-preorder traversal: all allot alloy aloe an ant are ate be

2. Клас **DictionaryImp2** за представяне на речник, съгласно спецификацията:

```

package Dictionary;

import java.util.Iterator;
import Interface.Dictionary;
import Function.Entry;
import PrefixTree.PrefixTree;

public class DictionaryImp2 extends PrefixTree implements Dictionary {
    //Конструктор
    public DictionaryImp2() { super(); }
}

```

```
//Методи, наследени от класа PrefixTree

//Реализация на методите на интерфейса Dictionary

//Предефиниране на наследени методи
public String toString() {...}

}
```

Реализация:

```
package Dictionary;

import java.util.Iterator;
import Interface.Dictionary;
import Function.Entry;
import PrefixTree.PrefixTree;

public class DictionaryImp2 extends PrefixTree implements Dictionary {
    // Constructor
    public DictionaryImp2() {
        super();
    }

    // Methods inherited from class PrefixTree

    // Dictionary methods implementation
    public boolean containsKey(String key) {
        return super.contains(key);
    }

    public boolean containsValue(Object value) {
        throw new UnsupportedOperationException();
    }

    public Object get(String key) {
        PrefixTree t = super.search(key);
        return t != null ? t.data() : null;
    }

    public Object put(String key, Object value) {
        PrefixTree t = super.search(key);
        if (t != null) {
            Object r = t.marker;
            t.marker = value;
            return r;
        }
        t = super.insert(key);
        t.marker = value;
        return null;
    }

    public Object remove(String key) {
        throw new UnsupportedOperationException();
    }

    public boolean equals(Object obj) {
        throw new UnsupportedOperationException();
    }

    @SuppressWarnings("unchecked")
    public Iterator keys() {
        return super.iterator();
    }

    @SuppressWarnings("unchecked")
```

```

public Iterator values() {
    throw new UnsupportedOperationException();
}

// Override
@SuppressWarnings("unchecked")
public String toString() {
    String result = "";
    Iterator it = keys();
    while (it.hasNext()) {
        String key = (String) it.next();
        Entry m = new Entry(key);
        m.setValue(get(key));
        result = result + m + "\n";
    }
    return result;
}
}

```

Тестване: Класът **TestDictionaryImp2** чете двойки от думи дума1@дума2 от текстов файл testDictData.txt и създава речник

Dictionary dict = new DictionaryImp2()

от двойки (дума1,дума2), като го съхранява в префиксно дърво с ключове първите думи, като за всеки ключ маркерът съдържа адреса на съответната втора дума. Обхожда дървото и извежда речника.

```

package Test;

import java.io.File;
import java.io.IOException;
import Dictionary.DictionaryImp2;
import Interface.Dictionary;

public class DictionaryImp2Tester {
    public static void main(String[] args) {
        try {
            String s = File.separator;
            String userDir = System.getProperty("user.dir");
            System.out.println("User directory: " + userDir);
            String dir = userDir + s + "src" + s + "Test";

            String source = dir + s + "testDictData.txt";
            System.out.println("Read text from the file " + source);

            Dictionary dict = new DictionaryImp2();
            new DictionaryTester().createDictionary(dict, source);
            System.out.println("      English-Bulgarian Dictionary:" + "\n"
                + dict);

        } catch (IOException e) {
        }
    }
}

```

Резултати от тестването:

```

User directory: D:\UserFolder\JavaProjects\SDP
Read text from the file D:\UserFolder\JavaProjects\SDP\src\Test\testDictData.txt

    Searching for string: black
Cannot find string: black
    Inserted characters:
b   l   a   c   k
Finished inserting the word: black

```

```
Searching for string: red
Cannot find string: red
    Inserted characters:
r   e   d
Finished inserting the word: red
```

```
Searching for string: white
Cannot find string: white
    Inserted characters:
w   h   i   t   e
Finished inserting the word: white
```

```
Searching for string: purple
Cannot find string: purple
    Inserted characters:
p   u   r   p   l   e
Finished inserting the word: purple
```

```
Searching for string: green
Cannot find string: green
    Inserted characters:
g   r   e   e   n
Finished inserting the word: green
```

```
Searching for string: blue
Found character: b
Found character: l
Cannot find string: blue
    Inserted characters:
b   l   u   e
Finished inserting the word: blue
```

```
Searching for string: yellow
Cannot find string: yellow
    Inserted characters:
y   e   l   l   o   w
Finished inserting the word: yellow
```

```
Searching for string: black
Found character: b
Found character: l
Found character: a
Found character: c
Found character: k
Found string: black
```

```
Searching for string: blue
Found character: b
Found character: l
Found character: u
Found character: e
Found string: blue
```

```
Searching for string: green
Found character: g
Found character: r
Found character: e
Found character: e
```

```
Found character: n
Found string: green

    Searching for string: purple
Found character: p
Found character: u
Found character: r
Found character: p
Found character: l
Found character: e
Found string: purple

    Searching for string: red
Found character: r
Found character: e
Found character: d
Found string: red

    Searching for string: white
Found character: w
Found character: h
Found character: i
Found character: t
Found character: e
Found string: white

    Searching for string: yellow
Found character: y
Found character: e
Found character: l
Found character: l
Found character: o
Found character: w
Found string: yellow
    English-Bulgarian Dictionary:
(black,4eren)
(blue,sin)
(green,zelen)
(purple,morav)
(red,4erven)
(white,bial)
(yellow,jalt)
```

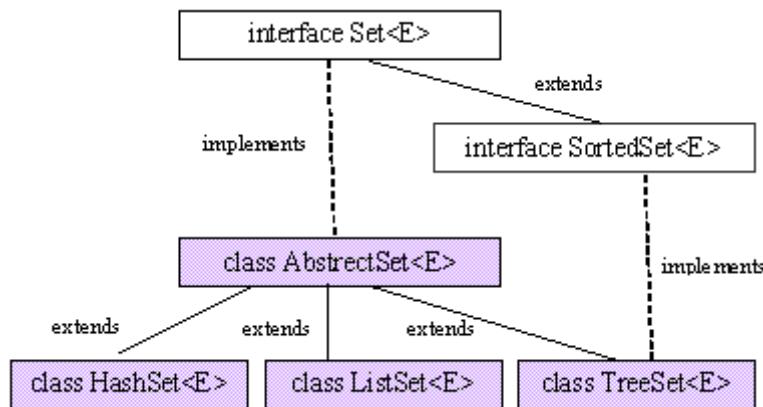
Файл testDictData.txt

```
black@4eren
red@4erven
white@bial
purple@morav
green@zelen
blue@sin
yellow@jalt
end
```



Множество

Задача: Да се реализират класовете от йерархията за представяне на множество:



Интерфейсът **Set<E>** представя операции на множество:

```

package Interface;

import java.util.Iterator;

public interface Set<E> {
    public void add(E e);
    // Добавя елемента е към текущото множество, ако той вече не се съдържа в
    // него

    public boolean addAll(Set<? extends E> arg);
    // Добавя към текущото множество несъдържащите се в него елементи на
    // множеството arg и връща true, ако текущото множество се е променило

    public void clear();
    // Премахва всички елементи на текущото множество

    public Object clone();
    //Създава и връща копие на текущото множество

    public boolean contains(E e);
    // Проверява дали елементът е се съдържа в текущото множество

    public boolean containsAll(Set<? extends E> arg);
    // Проверява дали текущото множество съдържа всички елементи на множеството
    // arg

    public boolean equals(Object arg);
    // Проверява дали текущото множество и множеството arg съвпадат

    public boolean isEmpty();
    // Проверява дали текущото множество е празно

    public Iterator<E> iterator();
    // Връща итератор за обхождане на елементите на текущото множество

    public boolean remove(E e);
    // Отстранява елемента е от текущото множество и връща true, ако такъв е
    // имало

    public boolean removeAll(Set<? extends E> arg);
    // Отстранява от текущото множество всички елементи, които се съдържат в
    // множеството arg и връща true, ако текущото множество се е променило
}

```

```

public boolean retainAll(Set<? extends E> arg);
// Оставя в текущото множеството само тези елементи, които принадлежат на
// множеството arg и връща true, ако текущото множество се е променило

public int size();
// Връща броя на елементите на текущото множество

public Object[] toArray();
// Връща масив, съдържащ елементите на текущото множество

public String toString();
// Връща низ от елементите на текущото множество
}

```

Интерфейсът **SortedSet<E>** представя операции на наредено множество:

```

package Interface;

public interface SortedSet<E> extends Set<E> {
    public Comparator<? super E> comparator();
    // Връща компаратор за сравняване на елементите на текущото множество, или
    // null, ако се използва NaturalComparator<E>

    public E first();
    // Връща най-малкия елемент на текущото сортирано множество

    public SortedSet<E> headSet(E toElement);
    // Създава и връща сортирано множество от елементите на текущото множество,
    // които са по-малки от елемента toElement

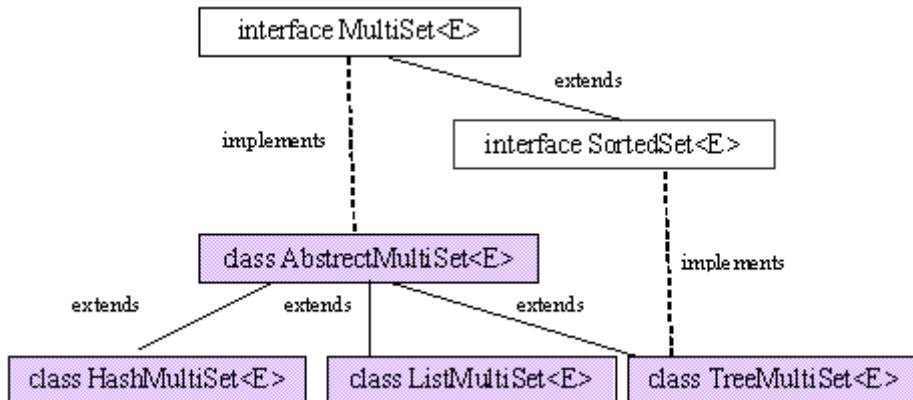
    public E last();
    // Връща най-големия елемент на текущото сортирано множество

    public SortedSet<E> subSet(E fromElement, E toElement);
    // Създава и връща сортирано множество от елементите на текущото множество,
    // които са по-големи или равни на елемента fromElement и по-малки от
    // елемента toElement

    public SortedSet<E> tailSet(E fromElement);
    // Създава и връща сортирано множество от елементите на текущото множество,
    // които са по-големи или равни на елемента fromElement
}

```

Задача: Да се реализират класовете от йерархията за представяне на мултимножество:



Мултимножество $M = \{(e_1, P(M, e_1)), (e_2, P(M, e_2)), \dots, (e_n, P(M, e_n))\}$ е множество от двойки, в които e_1, \dots, e_n са различните елементи на M , а $P(M, e_i) \geq 1$ е брой срещания на елемента e_i в M , $i \in [1; n]$. Ако $e \notin M$, $P(M, e) = 0$.

Операции: Нека А и В са мултимножества. Тогава:

- Обединение: $A \cup B = \{(e, P(A, e) + P(B, e)) \mid e \in A \text{ или } e \in B\}$
- Сечение: $A \cap B = \{(e, \min(P(A, e), P(B, e))) \mid e \in A \text{ и } e \in B\}$
- Разлика: $A \setminus B = \{(e, P(A, e) - \min(P(A, e), P(B, e))) \mid e \in A\}$
- Подмножество: $A \subseteq B$, ако за всеки елемент $e \in A$ е изпълнено $P(A, e) \leq P(B, e)$

Интерфейсът **Multiset<E>** съдържа операции на мултимножество:

```
import java.util.Iterator;

public interface Multiset<E> {
    public int put(E element, int p);
    // Включва p броя element в текущото мултимножество и връща брой срещания на
    // element след включването

    public int get(E element);
    // Връща брой срещания на element в текущото мултимножество

    public int remove(E element);
    // Извлича един брой element от текущото мултимножество и връща брой
    // срещания на element след изключването

    public void clear();
    // Премахва всички елементи на текущото множество

    public Object clone();
    // Създава и връща копие на текущото множество

    public boolean containsElement(E element);
    // Проверява дали element принадлежи на текущото мултимножество

    public int size();
    // Връща общия брой елементи на текущото мултимножество

    public boolean equals(Object arg);
    // Проверява дали текущото множество и множеството arg съвпадат

    public boolean isEmpty();
    // Проверява дали текущото мултимножество е празно

    public Iterator<E> iterator();
    // Обхожда различните елементи на текущото мултимножество

    public Multiset<E> union(Multiset<? extends E> m);
    // Определя обединението на текущото мултимножество и m

    public Multiset<E> intersection(Multiset<? extends E> m);
    // Определя сечението на текущото мултимножество и m

    public Multiset<E> difference(Multiset<? extends E> m);
    // Определя разликата на текущото мултимножество с m

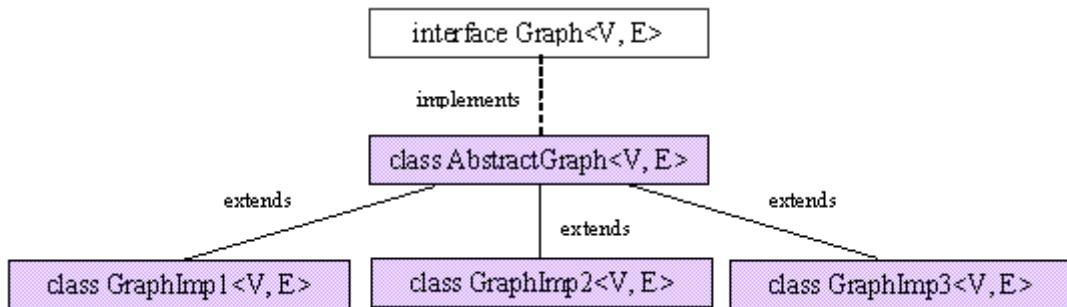
    public boolean subsetOf(Multiset<? extends E> m);
    // Проверява дали текущото мултимножество е подмножество на m

    public Object[] toArray();
    // Връща масив, съдържащ елементите на текущото множество

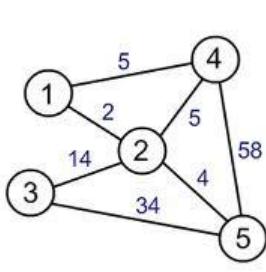
    public String toString();
    // Връща низ от елементите на текущото множество
}
```



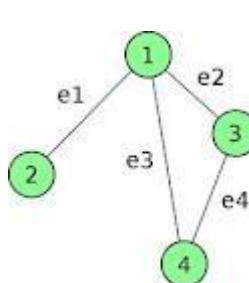
Задача: Дадена е юерархията от интерфейси и класове:



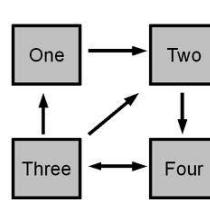
Интерфейсът **Graph<V, E>** представя операции на *ориентиран/неориентиран граф G(Vertices, Edges)* с множество от върхове *Vertices*, като всеки върх има уникално за графа име – стойност на типа *V* и множество от ребра *Edges*, като всяко ребро има етикет (тегло) – стойност на типа *E*, например:



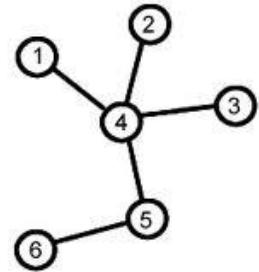
Неориентиран граф с върхове – цели числа и ребра с тегла – цели числа



Неориентиран граф с върхове – цели числа и ребра с тегла – низове



Ориентиран граф с върхове – низове и всяко ребро е с тегло 1



Неориентиран граф с върхове – цели числа и всяко ребро е с тегло 1

```

package Interface;

import java.util.Iterator;
import Graph.Edge;

public interface Graph<V, E extends Comparable<E>> {
    public V getNextVertex();
    //Връща недопустима стойност на типа V за върх на текущия граф

    public E getNotAnEdge();
    //Връща стойност на типа E за ребра без тегла на текущия граф

    public void add(V v);
    //Добавя върх v в текущия граф

    public void addEdge(V u, V v, E w);
    //Добавя ориентирано ребро от u до v с тегло w в текущия граф

    public V remove(V v);
    //Премахва върх v от текущия граф и инцидентните с него ребра

    public E removeEdge(V u, V v);
    //Премахва ориентирано ребро от u до v от текущия граф

    public Edge<V, E> getEdge(V u, V v);
    //Връща ориентирано ребро от u до v на текущия граф
  
```

```

public boolean contains(V v);
//Проверява дали v е връх на текущия граф

public boolean containsEdge(V u, V v);
//Проверява дали има ориентирано ребро от u до v в текущия граф

public int V();
//Връща броя на върховете на текущия граф

public int degree(V v);
//Връща броя на ориентираните ребра от връх v на текущия граф

public int E();
//Връща броя на ребрата на текущия граф

public Iterator<V> iterator();
//Връща итератор по върховете на текущия граф

public Iterator<V> neighborsIterator(V v);
//Връща итератор по съседните върхове на връх v на текущия граф

public Iterator<Edge<V, E>> edgesIterator();
//Връща итератор по ребрата на текущия граф

public String toString();
//Връща низ от данните на текущия граф

public boolean isDirected();
//Проверява дали текущият граф е ориентиран

public boolean isConnected();
//Проверява дали текущият граф е свързан

public boolean isEulerianGraph();
//Проверява дали текущият граф е Ойлеров - http://en.wikipedia.org/wiki/Eulerian\_path

public boolean isHamiltonianGraph();
//Проверява дали текущият граф е Хамилтонов - http://en.wikipedia.org/wiki/Hamiltonian\_path
}

```

Класът **Edge<V, E>** представя ориентирано ребро на граф (Начало, Край, Тегло):

```

package Graph;

public class Edge<V, E extends Comparable<E>> implements Comparable<Edge<V, E>>
{
    // Data
    private V begin, end;
    private E weight;

    // Constructor
    public Edge(V begin, V end, E weight) {
        this.begin = begin;
        this.end = end;
        this.weight = weight;
    }

    // Public methods
    public V getBegin() {
        return begin;
    }

    public V getEnd() {
        return end;
    }
}

```

```

public E getWeight() {
    return weight;
}

public int compareTo(Edge<V, E> arg) {
    if (this.weight.compareTo(arg.getWeight()) == 0)
        return 1;
    else
        return this.weight.compareTo(arg.getWeight());
}

// Override
@SuppressWarnings("unchecked")
public boolean equals(Object obj) {
    Edge<V, E> arg = (Edge<V, E>) obj;
    return begin.equals(arg.begin) && end.equals(arg.end);
}

public String toString() {
    return "(" + begin + ", " + end + ", " + weight + ")";
}
}

```

Да се реализират:

1. Клас **AbstractGraph<V, E>**, съгласно спецификацията:

```

package Graph;

import java.util.Iterator;
import Interface.Graph;

public abstract class AbstractGraph<V, E extends Comparable<E>> implements
    Graph<V, E> {
    //Данни
    protected V notAVertex;           //недопустима стойност на типа V за име на връх
    protected E notAnWeight;          //недопустима стойност на типа E за тегло на ребро

    // Конструктор
    public AbstractGraph(V notAVertex, E notAnWeight) { ... }

    //Абстрактни методи – от интерфейса Graph<V, E>

    //Реализация на методи на интерфейса Graph<V, E>
    public V getNotAVertex() { ... }
    public E getNotAnWeight() { ... }
    public boolean isDirected() { ... }
    public boolean isConnected() { ... }
    public boolean isEulerianGraph() { ... }
    public boolean isHamiltonianGraph() { ... }

    //Предефиниране на наследени методи от клас Object
    public String toString() { ... }
}

```

Реализация:

```

package Graph;

import java.util.Iterator;
import Interface.Graph;

public abstract class AbstractGraph<V, E extends Comparable<E>> implements
    Graph<V, E> {
    // Data

```

```

protected V notAVertex;
protected E notAnWeight;

// Constructors
public AbstractGraph(V notAVertex, E notAnWeight) {
    this.notAVertex = notAVertex;
    this.notAnWeight = notAnWeight;
}

// Public methods
public V getNotAVertex() {
    return notAVertex;
}

public E getNotAnWeight() {
    return notAnWeight;
}

public boolean isDirected() {
    throw new UnsupportedOperationException();
}

public boolean isConnected() {
    throw new UnsupportedOperationException();
}

public boolean isEulerianGraph() {
    throw new UnsupportedOperationException();
}

public boolean isHamiltonianGraph() {
    throw new UnsupportedOperationException();
}

public String toString() {
    String result = "";
    Iterator<V> it = iterator();
    while (it.hasNext()) {
        V u = it.next();
        result += "Vertex: " + u + "\nNeighbors: ";
        Iterator<V> it1 = neighborsIterator(u);
        while (it1.hasNext())
            result += it1.next() + " ";
        result += "\n";
    }
    return result;
}

```

2. Клас **GraphImp1<V, E>**, съгласно спецификацията:

```
package Graph;

import Interface.Graph;
import java.util.Map;
import java.util.Hashtable;
import java.util.HashSet;
import java.util.TreeSet;
import java.util.Iterator;

public class GraphImp1<V, E extends Comparable<E>> extends AbstractGraph<V, E>
        implements Graph<V, E> {
    //Данни
    private Map<V, TreeSet<Edge<V, E>>> table;
```

```

// Конструктор
public GraphImp1(V notAVertex, E notAnWeight) {...}

//Наследени методи от клас AbstractGraph<V, E>

//Реализация на методи на интерфейса Graph<V, E>
public void add(V v) {...}
public void addEdge(V u, V v, E w) {...}
public V remove(V v) {...}
public E removeEdge(V u, V v) {...}
public Edge<V, E> getEdge(V u, V v) {...}
public boolean contains(V v) {...}
public boolean containsEdge(V u, V v) {...}
public int V() {...}
public int degree(V v) {...}
public int E() {...}
public Iterator<V> iterator() {...}
public Iterator<V> neighborsIterator(V v) {...}
public Iterator<Edge<V, E>> edgesIterator() {...}
}

```

*Реализацията на класа **GraphImp1<V, E>** се основава на представяне на граф чрез таблица, като всеки елемент на таблицата е от вида*

(Връх, Множество_от_ребра,_инцидентни_с_върха,_сортирано_по_теглата)

```

package Graph;

import Interface.Graph;
import java.util.Map;
import java.util.Hashtable;
import java.util.HashSet;
import java.util.TreeSet;
import java.util.Iterator;

public class GraphImp1<V, E extends Comparable<E>> extends AbstractGraph<V, E>
    implements Graph<V, E> {
    // Data
    private Map<V, TreeSet<Edge<V, E>>> table;

    // Constructors
    public GraphImp1(V notAVertex, E notAnWeight) {
        super(notAVertex, notAnWeight);
        table = new Hashtable<V, TreeSet<Edge<V, E>>>();
    }

    // Public methods
    public void add(V v) {
        table.put(v, new TreeSet<Edge<V, E>>());
    }

    public void addEdge(V u, V v, E w) {
        table.get(u).add(new Edge<V, E>(u, v, w));
    }

    public V remove(V v) {
        table.remove(v);
        return v;
    }

    public E removeEdge(V u, V v) {
        E result = getEdge(u, v).getWeight();
        table.get(u).remove(new Edge<V, E>(u, v, notAnWeight));
        return result;
    }
}

```

```

public Edge<V, E> getEdge(V u, V v) {
    throw new UnsupportedOperationException();
}

public boolean contains(V v) {
    return table.containsKey(v);
}

public boolean containsEdge(V u, V v) {
    return table.get(u).contains(new Edge<V, E>(u, v, notAnWeight));
}

public int V() {
    return table.size();
}

public int degree(V v) {
    return table.get(v).size();
}

public int E() {
    throw new UnsupportedOperationException();
}

public Iterator<V> iterator() {
    return table.keySet().iterator();
}

public Iterator<V> neighborsIterator(V v) {
    Iterator<Edge<V, E>> it = table.get(v).iterator();
    HashSet<V> adj = new HashSet<V>();
    while (it.hasNext())
        adj.add(it.next().getEnd());
    return adj.iterator();
}

public Iterator<Edge<V, E>> edgesIterator() {
    TreeSet<Edge<V, E>> t = new TreeSet<Edge<V, E>>();
    Iterator<V> it = iterator();
    while (it.hasNext()) {
        Iterator<Edge<V, E>> it1 = table.get(it.next()).iterator();
        while (it1.hasNext())
            t.add(it1.next());
    }
    return t.iterator();
}

```

3. Клас **GraphImp2<V, E>**, съгласно спецификацията:

```
package Graph;

import Interface.Graph;

import java.util.Map;
import java.util.Hashtable;
import java.util.Iterator;

public class GraphImp2<V, E extends Comparable<E>> extends AbstractGraph<V, E>
        implements Graph<V, E> {
    //Данные
    private Map<V, Map<V, E>> table;

    // Конструктор
    public GraphImp2(V notAVertex, E notAnWeight) { ... }
```

```
//Наследени методи от клас AbstractGraph<V, E>

//Реализация на методи на интерфейса Graph<V, E>
public void add(V v) {...}
public void addEdge(V u, V v, E w) {...}
public V remove(V v) {...}
public E removeEdge(V u, V v) {...}
public Edge<V, E> getEdge(V u, V v) {...}
public boolean contains(V v) {...}
public boolean containsEdge(V u, V v) {...}
public int V() {...}
public int degree(V v) {...}
public int E() {...}
public Iterator<V> iterator() {...}
public Iterator<V> neighborsIterator(V v) {...}
public Iterator<Edge<V, E>> edgesIterator() {...}
}
```

Реализацията на класа GraphImp2<V, E> се основава на представяне на граф чрез таблица, като всеки елемент на таблицата е от вида (Връх_1, (Връх_2, Тегло)) и представлява ориентираното ребро (Връх_1, Връх_2, Тегло).

```
package Graph;

import Interface.Graph;

import java.util.Map;
import java.util.Hashtable;
import java.util.Iterator;

public class GraphImp2<V, E extends Comparable<E>> extends AbstractGraph<V, E>
    implements Graph<V, E> {
    // Data
    private Map<V, Map<V, E>> table;

    // Constructors
    public GraphImp2(V notAVertex, E notAnWeight) {
        super(notAVertex, notAnWeight);
        table = new Hashtable<V, Map<V, E>>();
    }

    // Public methods
    public void add(V v) {
        table.put(v, new Hashtable<V, E>());
    }

    public void addEdge(V u, V v, E w) {
        table.get(u).put(v, w);
    }

    public V remove(V v) {
        throw new UnsupportedOperationException();
    }

    public E removeEdge(V u, V v) {
        E result = getEdge(u, v).getWeight();
        table.get(u).remove(v);
        return result;
    }

    public Edge<V, E> getEdge(V u, V v) {
        throw new UnsupportedOperationException();
    }
}
```

```

public boolean contains(V v) {
    return table.containsKey(v);
}

public boolean containsEdge(V u, V v) {
    return table.get(u).containsKey(v);
}

public int V() {
    return table.size();
}

public int degree(V v) {
    return table.get(v).size();
}

public int E() {
    throw new UnsupportedOperationException();
}

public Iterator<V> iterator() {
    return table.keySet().iterator();
}

public Iterator<V> neighborsIterator(V v) {
    return table.get(v).keySet().iterator();
}

public Iterator<Edge<V, E>> edgesIterator() {
    throw new UnsupportedOperationException();
}
}

```

4. Клас GraphImp3<V, E>, съгласно спецификацията:

```

package Graph;

import Interface.Graph;
import java.util.PriorityQueue;
import java.util.Iterator;

public class GraphImp3<V, E extends Comparable<E>> extends AbstractGraph<V, E>
    implements Graph<V, E> {
    //Данни
    private PriorityQueue<Edge<V, E>> queue;

    // Конструктор
    public GraphImp3(V notAVertex, E notAnWeight) {...}

    //Наследени методи от клас AbstractGraph<V, E>

    //Реализация на методи на интерфейса Graph<V, E>
    public void add(V v) {...}
    public void addEdge(V u, V v, E w) {...}
    public V remove(V v) {...}
    public E removeEdge(V u, V v) {...}
    public Edge<V, E> getEdge(V u, V v) {...}
    public boolean contains(V v) {...}
    public boolean containsEdge(V u, V v) {...}
    public int V() {...}
    public int degree(V v) {...}
    public int E() {...}
    public Iterator<V> iterator() {...}
    public Iterator<V> neighborsIterator(V v) {...}
    public Iterator<Edge<V, E>> edgesIterator() {...}
}

```

}

Реализацията на класа **GraphImp2<V, E>** се основава на представяне на граф чрез **приоритетна опашка от ребрата на графа**.

```

package Graph;

import Interface.Graph;
import java.util.PriorityQueue;
import java.util.Iterator;

public class GraphImp3<V, E extends Comparable<E>> extends AbstractGraph<V, E>
    implements Graph<V, E> {
    // Data
    private PriorityQueue<Edge<V, E>> queue;

    // Constructors
    public GraphImp3(V notAVertex, E notAnWeight) {
        super(notAVertex, notAnWeight);
        queue = new PriorityQueue<Edge<V, E>>();
    }

    // Public methods
    public void add(V v) {
        throw new UnsupportedOperationException();
    }

    public void addEdge(V u, V v, E w) {
        queue.add(new Edge<V, E>(u, v, w));
    }

    public V remove(V v) {
        throw new UnsupportedOperationException();
    }

    public E removeEdge(V u, V v) {
        E result = getEdge(u, v).getWeight();
        queue.remove(new Edge<V, E>(u, v, notAnWeight));
        return result;
    }

    public Edge<V, E> getEdge(V u, V v) {
        throw new UnsupportedOperationException();
    }

    public boolean contains(V v) {
        throw new UnsupportedOperationException();
    }

    public boolean containsEdge(V u, V v) {
        return queue.contains(new Edge<V, E>(u, v, notAnWeight));
    }

    public int V() {
        throw new UnsupportedOperationException();
    }

    public int degree(V v) {
        throw new UnsupportedOperationException();
    }

    public int E() {
        return queue.size();
    }
}

```

```

public Iterator<V> iterator() {
    throw new UnsupportedOperationException();
}

public Iterator<V> neighborsIterator(V v) {
    throw new UnsupportedOperationException();
}

public Iterator<Edge<V, E>> edgesIterator() {
    return queue.iterator();
}
}

```

Приложения:

1. Клас **GraphTraversalObject<V, E>**

Реализация:

```

package Graph;

import Interface.Graph;
import java.util.Iterator;
import java.util.Map;
import java.util.Hashtable;

public class GraphTraversalObject<V, E extends Comparable<E>> {
    // Data
    protected V notAVertex;
    protected E notAnWeight;
    protected Map<V, Boolean> visited;

    // Constructor
    public GraphTraversalObject(Graph<V, E> g, V v) {
        notAVertex = g.getNotAVertex();
        notAnWeight = g.getNotAnWeight();
        visited = new Hashtable<V, Boolean>();
        Iterator<V> it = g.iterator();
        while (it.hasNext())
            visited.put(it.next(), false);
    }
}

```

2. Клас **DepthFirstSearcher<V, E>** за проверка дали има път между два върха в граф, като графът се обхожда в дълбочина

Реализация:

```

package Graph;

import Interface.Graph;
import java.util.Iterator;

public class DepthFirstSearcher<V, E extends Comparable<E>> extends
    GraphTraversalObject<V, E> {
    // Constructor
    public DepthFirstSearcher(Graph<V, E> g, V v) {
        super(g, v);
        search(g, v);
    }

    // Private method
    private void search(Graph<V, E> g, V v) {
        visited.put(v, true);
        Iterator<V> it = g.neighborsIterator(v);

```

```

while (it.hasNext()) {
    V w = it.next();
    if (!visited.get(w))
        search(g, w);
}
}

// Public method
public boolean isConnected(V w) {
    return visited.get(w);
}
}
}

```

3. Клас **DepthFirstPathSearcher<V, E>** за определяне на път между два върха в граф, като графът се обхожда в дълбочина

Реализация:

```

package Graph;

import Interface.Graph;
import java.util.Map;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;

public class DepthFirstPathSearcher<V, E extends Comparable<E>> extends
    GraphTraversalObject<V, E> {
    // Data
    private Map<V, V> parents;

    // Constructor
    public DepthFirstPathSearcher(Graph<V, E> g, V v) {
        super(g, v);
        parents = new Hashtable<V, V>();
        Iterator<V> it = g.iterator();
        while (it.hasNext())
            parents.put(it.next(), notAVertex);
        search(g, v);
    }

    // Private method
    private void search(Graph<V, E> g, V v) {
        visited.put(v, true);
        Iterator<V> it = g.neighborsIterator(v);
        while (it.hasNext()) {
            V w = it.next();
            if (!visited.get(w)) {
                parents.put(w, v);
                search(g, w);
            }
        }
    }

    // Public method
    public Iterable<V> getPathTo(V w) {
        LinkedList<V> path = new LinkedList<V>();
        while (!w.equals(notAVertex) && visited.get(w)) {
            path.addFirst(w);
            w = parents.get(w);
        }
        return path;
    }
}

```

4. Клас **DepthFirstTreeCreator<V, E>** за построяване на покриващо дърво на граф, като графът се обхожда в дълбочина

Реализация:

```
package Graph;

import Interface.Graph;
import java.util.Iterator;

public class DepthFirstTreeCreator<V, E extends Comparable<E>> extends
    GraphTraversalObject<V, E> {
    // Data
    private Graph<V, E> tree;

    // Constructor
    public DepthFirstTreeCreator(Graph<V, E> g, V v) {
        super(g, v);
        tree = new GraphImpl<V, E>(notAVertex, notAnWeight);
        search(g, v);
    }

    // Private method
    private void search(Graph<V, E> g, V v) {
        tree.add(v);
        visited.put(v, true);
        Iterator<V> it = g.neighborsIterator(v);
        while (it.hasNext()) {
            V w = it.next();
            if (!visited.get(w)) {
                tree.addEdge(v, w, notAnWeight);
                search(g, w);
            }
        }
    }

    // Public method
    public Graph<V, E> getTree() {
        return tree;
    }
}
```

5. Клас **BreadthFirstMinPathSearcher<V, E>** за определяне на път с минимална дължина (брой ребра) между два върха в граф, като графът се обхожда в ширина

Реализация:

```
package Graph;

import Interface.Graph;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Map;

public class BreadthFirstMinPathSearcher<V, E extends Comparable<E>> extends
    GraphTraversalObject<V, E> {
    // Data
    private Map<V, Integer> table;

    // Constructor
    public BreadthFirstMinPathSearcher(Graph<V, E> g, V v) {
        super(g, v);
        table = new Hashtable<V, Integer>();
```

```

int n = g.V() + 1;
Iterator<V> it = g.iterator();
while (it.hasNext())
    table.put(it.next(), n);
table.put(v, 0);
search(g, v);
}

// Private method
private void search(Graph<V, E> g, V v) {
    LinkedList<V> q = new LinkedList<V>();
    q.addLast(v);
    while (!q.isEmpty()) {
        V u = q.removeFirst();
        Iterator<V> it = g.neighborsIterator(u);
        while (it.hasNext()) {
            V w = it.next();
            if (table.get(w) > g.V()) {
                q.addLast(w);
                table.put(w, table.get(u) + 1);
            }
        }
    }
}

// Public method
public int getMinDistanceTo(V w) {
    return table.get(w);
}
}

```

6. Клас **BreadthFirstTreeCreator<V, E>** за построяване на покриващо дърво на граф, като графът се обхожда в ширина

Реализация:

```

package Graph;

import Interface.Graph;
import java.util.Iterator;
import java.util.LinkedList;

public class BreadthFirstTreeCreator<V, E extends Comparable<E>> extends
    GraphTraversalObject<V, E> {
    // Data
    private Graph<V, E> tree;

    // Constructor
    public BreadthFirstTreeCreator(Graph<V, E> g, V v) {
        super(g, v);
        tree = new GraphImpl<V, E>(notAVertex, notAnWeight);
        search(g, v);
    }

    // Private method
    private void search(Graph<V, E> g, V v) {
        LinkedList<V> q = new LinkedList<V>();
        q.addLast(v);
        tree.add(v);
        visited.put(v, true);
        while (!q.isEmpty()) {
            V u = q.removeFirst();
            Iterator<V> it = g.neighborsIterator(u);
            while (it.hasNext()) {
                V w = it.next();

```

```

        if (!visited.get(w)) {
            tree.add(w);
            visited.put(w, true);
            tree.addEdge(u, w, notAnWeight);
            q.addLast(w);
        }
    }
}

// Public method
public Graph<V, E> getTree() {
    return tree;
}
}
}

```

Тестване: Клас **GraphTraversalObjectTester**

```

package Test;

import Interface.Graph;
import Graph.BreadthFirstMinPathSearcher;
import Graph.BreadthFirstTreeCreator;
import Graph.Edge;
import Graph.GraphImpl;
import Graph.DepthFirstSearcher;
import Graph.DepthFirstPathSearcher;
import Graph.DepthFirstTreeCreator;
import java.util.Iterator;

public class GraphTraversalObjectTester {
    private void test() {
        System.out.println("      Testing...");

        Graph<Integer, Integer> g = new GraphImpl<Integer, Integer>(-1, 0);
        for (int i = 1; i <= 8; i++)
            g.add(i);
        g.addEdge(1, 2, 1);
        g.addEdge(2, 1, 1);
        g.addEdge(1, 4, 1);
        g.addEdge(4, 1, 1);
        g.addEdge(2, 3, 1);
        g.addEdge(3, 2, 1);
        g.addEdge(4, 2, 1);
        g.addEdge(2, 4, 1);
        g.addEdge(5, 2, 1);
        g.addEdge(2, 5, 1);
        g.addEdge(6, 2, 1);
        g.addEdge(2, 6, 1);
        g.addEdge(5, 4, 1);
        g.addEdge(4, 5, 1);
        g.addEdge(4, 7, 1);
        g.addEdge(7, 4, 1);
        g.addEdge(5, 6, 1);
        g.addEdge(6, 5, 1);
        g.addEdge(7, 6, 1);
        g.addEdge(6, 7, 1);
        g.addEdge(8, 6, 1);
        g.addEdge(6, 8, 1);
        g.addEdge(8, 7, 1);
        g.addEdge(7, 8, 1);
        System.out.println("Graph g:\n" + g);
        System.out.println("Graph g edges:");
        Iterator<Edge<Integer, Integer>> it = g.edgesIterator();
        while (it.hasNext())
            System.out.println(it.next());
    }
}
}

```

```

        System.out.print("\nDepth first searcher: path(1,8) -> ");
        DepthFirstSearcher<Integer, Integer> s = new
DepthFirstSearcher<Integer, Integer>(
            g, 1);
        boolean flag = s.isConnected(8);
        System.out.println(flag ? "Yes" : "No");

        System.out.print("\nDepth first path searcher: path(1,8) -> ");
        DepthFirstPathSearcher<Integer, Integer> s2 = new
DepthFirstPathSearcher<Integer, Integer>(
            g, 1);
        for (Integer u : s2.getPathTo(8))
            System.out.print(u + "   ");
        System.out.println();

        System.out.println("\nDepth first tree creator:");
        DepthFirstTreeCreator<Integer, Integer> s3 = new
DepthFirstTreeCreator<Integer, Integer>(
            g, 1);
        Graph<Integer, Integer> t = s3.getTree();
        System.out.println("Tree t: root = 1\n" + t);

        BreadthFirstMinPathSearcher<Integer, Integer> s4 = new
BreadthFirstMinPathSearcher<Integer, Integer>(
            g, 1);
        System.out.println("\nBreadth first min path searcher: path(1,8) ->
"
                + s4.getMinDistanceTo(8));

        System.out.println("\nBreadth first tree creator:");
        BreadthFirstTreeCreator<Integer, Integer> s5 = new
BreadthFirstTreeCreator<Integer, Integer>(
            g, 1);
        t = s5.getTree();
        System.out.println("Tree t: root = 1\n" + t);

        System.out.println("Done!");
    }

    public void start() {
        System.out.println("      Test");
        test();
    }

    public static void main(String[] args) {
        new GraphTraversalObjectTester().start();
    }
}

```

Резултати от тестването:

```

Test
Testing...
Graph g:
Vertex: 8
Neighbors: 6   7
Vertex: 7
Neighbors: 4   6   8
Vertex: 6
Neighbors: 2   5   7   8
Vertex: 5
Neighbors: 2   4   6
Vertex: 4
Neighbors: 1   2   5   7
Vertex: 3

```

```
Neighbors: 2
Vertex: 2
Neighbors: 1 3 4 5 6
Vertex: 1
Neighbors: 2 4
```

Graph g edges:

```
(8, 6, 1)
(8, 7, 1)
(7, 4, 1)
(7, 6, 1)
(7, 8, 1)
(6, 2, 1)
(6, 5, 1)
(6, 7, 1)
(6, 8, 1)
(5, 2, 1)
(5, 4, 1)
(5, 6, 1)
(4, 1, 1)
(4, 2, 1)
(4, 5, 1)
(4, 7, 1)
(3, 2, 1)
(2, 1, 1)
(2, 3, 1)
(2, 4, 1)
(2, 5, 1)
(2, 6, 1)
(1, 2, 1)
(1, 4, 1)
```

```
Depth first searcher: path(1,8) -> Yes
```

```
Depth first path searcher: path(1,8) -> 1 2 4 5 6 7 8
```

Depth first tree creator:

```
Tree t: root = 1
Vertex: 8
Neighbors:
Vertex: 7
Neighbors: 8
Vertex: 6
Neighbors: 7
Vertex: 5
Neighbors: 6
Vertex: 4
Neighbors: 5
Vertex: 3
Neighbors:
Vertex: 2
Neighbors: 3 4
Vertex: 1
Neighbors: 2
```

```
Breadth first min path searcher: path(1,8) -> 3
```

Breadth first tree creator:

```
Tree t: root = 1
Vertex: 8
Neighbors:
Vertex: 7
Neighbors:
Vertex: 6
Neighbors: 8
```

```

Vertex: 5
Neighbors:
Vertex: 4
Neighbors: 7
Vertex: 3
Neighbors:
Vertex: 2
Neighbors: 3 5 6
Vertex: 1
Neighbors: 2 4

```

Done!



Задача: Да се състави програма, която за ориентиран граф $G(V, E)$:

- Определя дължината на най-късия път между два дадени негови върха
- Определя всички върхове на графа, недостижими от даден негов връх
- Определя дължината на най-късия цикъл в графа
- Проверява дали графът е свързан (между всеки два върха има път в графа)
- Проверява дали графът е дърво (свързан граф без цикли)

Упътване: Нека M е матрицата на съседства на графа $G(V, E)$ и $M^k = \|a_{ij}^{(k)}\|$ е k -та степен на M . Тогава $a_{ij}^{(k)}$ е броят на пътищата с дължина k от връх v_i до връх v_j при $i \neq j$ и броят на циклите при $i = j$.

Задача: Да се състави клас, който съдържа:

- Метод, който проверява има ли път между два града от пътна карта
- Метод, който намира произволен път между два града от пътна карта
- Метод, който намира всички пътища между два града от пътна карта
- Метод, който намира път с минимална дължина между два града от пътна карта

Упътване: Пътна карта, съдържаща N града, може да се представи като граф $G(V, E)$, където V е множеството от градове и за всеки два града t_i и t_j , между които има пряк път, $(t_i, t_j) \in E$. Нека $M = \|a_{ij}\|$ с размери $N \times N$ е матрицата на съседства на графа $G(V, E)$, като $a_{ij} = a_{ji} = 1$, ако има пряк път между t_i и t_j и 0 в противен случай. Намирането на път между два града А и В се свежда до намиране на пряк път от А до междинен град С и намиране на път от С до В:

1. Нека $p = 0$
2. Ако $A = B$ или A и B са пряко свързани, тогава $p = 1$. В противен случай преход на т.3
3. Ако има град C , пряко свързан с A , то:
 - 3.1.Премахване на проката връзка между A и C
 - 3.2.Намиране на път между C и B (рекурсивно обръщение)
 - 3.3.Възстановяване на проката връзка между A и C
4. Ако $p = 1$ или няма повече градове, пряко свързани с A , преход на т.5. В противен случай преход на т.3
5. Край

Литература

1. Cole J. H. Foster, Using Moodle, 2ed Edition, O’Raiilly, 2007
2. Goodrich M., R. Tamassia, Data Structures and Algorithms in Java, Wiley, 1998
3. Gosling J., B. Joy, G. Steele, G. Bracha, The Java Language Specification, Second Edition, 2000 Sun Microsystems, Inc. [The Java Language Specification, Third Edition](#)
4. Horstmann C., Big Java, 2ed Edition, Wiley, 2005
5. Preiss B., Data Structures and Algorithms with Object-Oriented Design Patterns in Java, Wiley, 1999
6. The Java Tutorial, <http://java.sun.com/docs/books/tutorial>
7. The Java Virtual Machine Specification, Second Edition, Addison-Wesley, 1999, http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html
8. Wikipedia, the free encyclopedia <http://www.wikipedia.org/>
9. Wilkie G., Object-Oriented Software Engineering. The Professional Developer’s Guide, Addison-Wesley, 1993
10. Wirth N., Algorithms and Data Structures, Prentice Hall, 1985.
11. Златопольский Д. М., Сборник задач по программированию. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2007. – 240 с.: ил.
12. Кнут Д., Искусство программирования для ЭВМ, т. 1, Основные алгоритмы, Пер. с англ., М.: Мир, 1976
13. Кнут Д., Искусство программирования для ЭВМ, т. 2, Получисленные алгоритмы, Пер. с англ., М.: Мир, 1977
14. Кнут Д., Искусство программирования для ЭВМ, т. 3, Сортировка и поиск, Пер. с англ., М.: Мир, 1978
15. Манев К., Увод в дискретната математика, Издателство на НБУ, 1996
16. Стоун Кр. и Дж. Уебър, Тайните на Java. Програмиране за Интернет, Книга първа, LIO Book Publishing, София 1997
17. Стоун Кр. и Дж. Уебър, Тайните на Java. Програмиране за Интернет, Книга втора, LIO Book Publishing, София 1997
18. Транбле Ж., П. Соренсон, Введение в структуры данных, Пер. с англ., М.: Машиностроение, 1982
19. Хорстман К., Принципи на програмирането с Java, Пер. с англ., ИК Софтех, София 2000