ლ. გაჩეჩილაძე, ლ. ნონიკაშვილი

ობიექტზე ორიენტირებული დაპროგრამება

Java ენაზე

I ნაწილი

მეთოდური მითითებები

ლაბორატორიული სამუშაოების შესასრულებლად

თბილისი

2014

განხილულია 15 ლაბორატორიული სამუშაოს შესრულების მეთოდიკა. ყოველი სამუშაოსთვის წარმოდგენილია თეორიული წანამძღვრები და აღწერილია პროგრამული კოდის ჩაწერისა და მისი შესრულების პროცედურები.

მოცემული მეთოდური მითითებანი განკუთვნილია Java ენაზე ობიექტზე ორიენტირებული დაპროგრამების ასათვისებლად.

რეცენზენტი პროფ. ო. კოტრიკაძე

ISBN 978-9941-0-7009-9 (ორივე ნაწილის)

ISBN 978-9941-0-7010-5 (პირველი ნაწილის)

შესავალი

Java ზოგადი დანიშნულების, ამავდროულად, მკაცრად (სტატიკურად) ტიპიზირებული, ობიექტზე ორიენტირებული დაპროგრამების ენაა. სინტაქსური აგებულებით ის წაგავს უფრო ადრეულ ენებს C და C++. მისი ობიექტური მოდელი მნიშვნელოვნად ნასესხებია C++ და Smalltalk ენებიდან.

Java აგებულია, როგორც ექსკლუზიურად ობიექტზე ორიენტირებული ენა. მთელი საწყისი კოდი იწერება კლასის შიგნით. ამიტომ პრიმიტიული ტიპების გარდა Java-ში ყველაფერი ობიექტია.

Java ენისაგან განსხვავებით, C და C++ ენებში პროგრამის საწყისი კოდის კომპილაცია ხორციელდება შესრულებად კოდში, რომელიც დაკავშირებულია კონკრეტულ პროცესორთან და ოპერაციულ სისტემასთან. სხვადასხვა პლატფორმაზე ასეთი პროგრამის გასაშვებად აუცილებელია პროგრამის საწყისი კოდის კომპილირება თითოეული პლატფორმისათვის. ეს კი შრომატევადი და ძვირადღირებული პროცესია.

Java ენაზე დაწერილი პროგრამის გადატანითობა მიიღწევა პროგრამის საწყისი კოდის შუალედურ ენაში ტრანსლირების გზით, რომელსაც ბაიტ-კოდი ეწოდება. შემდეგ ეს

შუალედური ენა სრულდება Java ვირტუალური მანქანის მიერ (Java Virtual Machine, JVM). შედეგად, Java-პროგრამა შეიძლება შესრულდეს ნებისმიერ პლატფორმაზე, რომელსაც Java ვირტუალური მანქანა აქვს.

Java ვირტუალური მანქანის კოდი (bytecode) წარმოადგენს უმაღლეს დონეზე ოპტიმიზირებული ინსტრუქციების ნაკრებს, რომელიც განკუთვნილია სისტემისთვის.

მნიშვნელოვანია იმის გაგებაც, რომ მთელი Java-პროგრამის ერთდროული კომპილაცია შესრულებად კოდში, მიზანშეწონილი არ არის, რადგან Java ასრულებს სხვადასხვა სახის შემოწმებებს, რომლებიც ხორციელდება მხოლოდ პროგრამის შესრულებაზე გაშვების დროს. ამასთან, კომპილაციას განიცდის ვირტუალური მანქანის კოდის არა ყველა ფრაგმენტი, არამედ მხოლოდ ის ნაწილი, რომლის კომპილაციაც მნიშვნელოვანია. კოდის დარჩენილი ნაწილი კი მხოლოდ ინტერპრეტირდება.

ობიექტზე ორიენტირებულობის კონცეფციები საფუძვლად უდევს Java ენას. მთავარია, გვესმოდეს, თუ როგორ ხდება ამ კონცეფციების პროგრამებში რეალიზება.

ლაბორატორიული სამუშაო №1

Java ენის ლექსიკა. პირველი პროგრამა

ტრადიციის მიხედვით, მრავალი სახელმძღვანელო იწყება უმარტივესი პროგრამით “Hello World!”. ქვემოთ წარმოდგენილია Java-ზე დაწერილი ეს პროგრამა.

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

ამ მარტივ მაგალითზეც შესაძლებელია შევამჩნიოთ ენის მთელი რიგი თავისებურებები:

- ყოველი პროგრამა წარმოადგენს ერთ ან რამოდენიმე **კლასს**. ჩვენ მაგალითში მხოლოდ ერთი კლასია (**class**).
- კლასის დასაწყისს აღნიშნავს დარეზერვებული სიტყვა **class**, რომელსაც მოსდევს პროგრამისტის მიერ შერჩეული კლასის კონკრეტული სახელი, ჩვენ შემთხვევაში **HelloWorld**. ყველაფერი, რაც კლასს ეკუთვნის, ფიგურულ ფრჩხილებში იწერება და წარმოადგენს კლასის ტანს (**class body**).

- ყოველი მოქმედება (გამოთვლები) წარმოებს ინფორმაციის დამუშავების მეთოდების (**methods**) საშუალებით.
- მეთოდებს აქვთ განსხვავებული სახელები. ერთ-ერთი მეთოდის სახელი აუცილებლად უნდა იყოს **main**. ამ მეთოდით იწყება პროგრამის შესრულება. ჩვენ მარტივ მაგალითში ერთი მეთოდია, ამიტომ მისი სახელია **main**.
- როგორც წესი, მეთოდი შესრულების შედეგად იძლევა (ხშირად ვამბობთ: აბრუნებს **return**) ერთ მნიშვნელობას, რომლის ტიპი მეთოდის სახელის წინ მიეთითება. მეთოდმა შეიძლება არაფერი არ დააბრუნოს, როდესაც იგი პროცედურის როლს ასრულებს, როგორც ეს ჩვენ შემთხვევაშია. ასეთ დროს დაბრუნებული მნიშვნელობის ტიპის მაგივრად იწერება სიტყვა **void**.
- მეთოდის სახელის შემდეგ ფრჩხილებში თავსდება არგუმენტების ან პარამეტრების ჩამონათვალი, რომლებიც ერთმანეთისაგან მძიმეებით გამოიყოფა. ყოველი არგუმენტისათვის ეთითება მისი ტიპი და ცარიელი სიმბოლოს შემდეგ - სახელი. მაგალითში მხოლოდ ერთი არგუმენტია, ის სტრიქონული მასივის ტიპისაა. სიმბოლოების მასივი - ესაა Java-ში არსებული String

ტიპი. კვადრატული ფრჩხილები მიუთითებს მასივს. მასივის სახელი ნებისმიერი დასაშვები იდენტიფიკატორი შეიძლება იყოს. მაგალითში არჩეულია **args**.

- მეთოდის დაბრუნებული ტიპის წინ შეიძლება ჩაწერილ იქნას სხვადასხვა **მოდიფიკატორი** (**modifiers**). მაგალითში ორი მოდიფიკატორია: სიტყვა **public** აღნიშნავს, რომ ეს მეთოდი წვდომადია ყველასათვის; სიტყვა **static** უზრუნველყოფს **main()** მეთოდის გამოძახების შესაძლებლობას პროგრამის შესრულების დასაწყისში. ზოგადად, მოდიფიკატორები არაა აუცილებელი, მაგრამ **main()** მეთოდისათვის ისინი სავალდებულოა. მეთოდის სახელის შემდეგ ყოველთვის ვწერთ ფრჩხილებს, რითაც ხაზს ვუსვამთ, რომ ეს მეთოდის სახელია და არა ჩვეულებრივი ცვლადის.
- მეთოდის მთელი შემცველობა წარმოადგენს მეთოდის ტანს (**method body**) და იწერება ფიგურულ ფრჩხილებში.

ერთადერთი მოქმედება, რაც **main()** მეთოდში ხდება, ესაა სხვა მეთოდის გამოძახება, რომელსაც რთული სახელი აქვს **System.out.println()** და მას გადაეცემა ერთი არგუმენტი, ტექსტური კონსტანტა **„Hello World!“.** ტექსტური კონსტანტა იწერება ორმაგ ბრჭყალებში, რაც გამყოფს წარ-

მოადგენს და ტექსტის შემადგენლობაში არ შედის. **println()** მეთოდი თავის არგუმენტს გაიტანს გამავალ ნაკადში, რომელიც, ჩვეულებრივ, ტექსტური ტერმინალის ეკრანთანაა დაკავშირებული. ტექსტის გამოტანის შემდეგ, კურსორი გადადის შემდეგი სტრიქონის დასაწყისში (ამას მიუთითებს დაბოლოება **ln**, სიტყვა **println –** შემოკლებით **print line**).

მნიშვნელოვანია აღინიშნოს, რომ Java-ს კომპილატორი განასხვავებს დიდ და პატარა ასოებს. პროგრამაში სიტყვები **String, System** უნდა დაიწყოს დიდი ასოებით, ხოლო **main**-პატარა ასოთი. ტექსტური კონსტანტის შიგნით კი მნიშვნელოვანი არაა პატარა ასოები იქნება გამოყენებული თუ დიდი, განსხვავება მხოლოდ ეკრანზე გამოჩნდება.

Java-ში პროგრამისტის მიერ შერჩეული სახელები შეიძლება ჩაიწეროს მისი შეხედულებისამებრ. კლასისთვის შეგვეძლო დაგვერქვა: **helloworld** ან **helloWorld**, მაგრამ Java-პროგრამისტებს შორის მოქმედებს შეთანხმება, რომელსაც უწოდებენ „**Code Conventions for the Java Programming Language**“, რომელსაც შეიძლება გაეცნოთ მისამართზე: **java.sun.com/docs/codeconv/index.html**. ამ შეთანხმების ზოგიერთი პუნქტი ასეთია:

- კლასის სახელები იწყება დიდი ასოებით; თუ სახელი შედგება რამდენიმე სიტყვისაგან, ყოველი შემდეგი სიტყვა დიდი ასოთი უნდა იწყებოდეს;
- ცვლადებისა და მეთოდების სახელები უნდა იწყებოდეს პატარა ასოთი; თუ სახელი რამდენიმე სიტყვისაგან შედგება, ყოველი შემდეგი სიტყვა დიდი ასოთი უნდა იწყებოდეს;
- კონსტანტების სახელები იწერება მხოლოდ დიდი ასოებით; თუ სახელი რამდენიმე სიტყვისაგან შედგება, ყოველ სიტყვას შორის ხაზგასმის სიმბოლო უნდა ჩაიწეროს;

ამ წესების გათვალისწინება აუცილებელი არ არის, მაგრამ მათი დაცვა მნიშვნელოვნად აადვილებს პროგრამის კოდის კითხვადობას და მას Java-ს სტილს აძლევს.

Java თავისუფალი ფორმატის ენაა. ეს ნიშნავს, რომ პროგრამის დაწერისას რაიმე სპეციალური წესების დაცვა აბზაცების მიმართ საჭირო არ არის. ერთადერთი, აუცილებელი მოთხოვნაა, რომ ყოველ ლექსემას შორის, რომლებიც ოპერაციის სიმბოლოთი ან გამყოფით გამოყოფილი არაა, ჩასმული იყოს ერთი ცარიელი სიმბოლო მაინც.

Java-ში ცარიელ სიმბოლოდ ითვლება ჰარი (**space**), ტაბულაცია ან ახალი სტრიქონის სიმბოლო.

დავალება

1. Java-დაპროგრამების ენაზე წარმოადგინეთ შემდეგი ტექსტის ამსახველი პროგრამა:

   „მეცნიერებმა შექმნეს ვირუსული პროგრამების ახალი სახეობა, რომელიც კავშირს მოწყობილობებს შორის ადამიანის ყურისთვის მიუწვდომელი სიგნალებით ახორციელებს. ასე და ამგვარად, მოწყობილობა ცვლის პაროლებს და სხვა პირად ინფორმაციას. ვირუსი მხოლოდ ჩაშენებულ მიკროფონებსა და დინამიკებს იყენებს მობილურ მოწყობილობებში. სპეციალისტებმა შეძლეს გადაეცათ პაროლები და სხვა მცირე მოცულობის ინფორმაცია 20 მეტრის მანძილზე“.

2. Java-დაპროგრამების ენაზე ჩაწერეთ ვარსკვლავებით ნაჩვენები მართკუთხა სამკუთხედის ამსახველი პროგრამა:

   *
   * *
   * * *
   * * * *
   * * * * *

ლაბორატორიული სამუშაო №2

Java ენის ბაზისური ტიპები და ოპერაციები მათზე

Java დაპროგრამების ენაში მრავალი მონაცემთა ტიპი და მათი გამოყენების კიდევ უფრო მეტი წესი არსებობს. ამ წესების დაუცველობა იწვევს დაფარულ შეცდომებს, რომელთა აღმოჩენაც საკმაოდ ძნელი და შრომატევადი სამუშაოა.

საწყისი მონაცემების ყველა ტიპი, რომლის აღწერაც შესაძლებელია Java-ში, იყოფა ორ ჯგუფად: პრიმიტიული (ბაზისური) (**primitive types**) და აღწერითი (**reference types** მიმთითებელი, მისამართი, მაკავშირებელი) ტიპები.

აღწერითი ტიპები იყოფა მასივებად (**arrays**), კლასებად (**classes**) და ინტერფეისებად (**interfaces**).

პრიმიტიული სულ 8 ტიპია. ისინი შეიძლება დაიყოს **რიცხვით** (**numeric**), **სიმბოლურ** (**character**) და **ლოგიკურ** (**boolean** ბულის) ტიპებად.

რიცხვითი ტიპები თავის მხრივ იყოფა: მთელრიცხვა (**integer**) და ნამდვილ (**floating-point**) ტიპებად:

- **მთელრიცხვა.** ამ ჯგუფში შედის **byte, short, int, long** ტიპები, რომლებიც წარმოადგენენ ნიშნიან მთელ რიცხვებს;

- **ნამდვილი (მცოცავმძიმიანი) რიცხვები.** ამ ჯგუფში შედის **float, double** ტიპები, ისინი წარმოადგენენ ნიშნიან წილად რიცხვებს, რომლებიც ათობითი წერტილის შემდეგ თანრიგების გარკვეული სიზუსტით მოიცემა;
- **სიმბოლოების** ტიპია **char**, რომელიც წარმოადგენს ენაში დაშვებულ ყველა სიმბოლოს, მაგალითად, ასოებს, ციფრებს, სასვენ ნიშნებს, სპეციალურ სიმბოლოებს და სხვა. ხშირად ამ ტიპს მიაკუთვნებენ მთელრიცხვა ტიპს და განიხილავენ, როგორც უნიშნო მთელ რიცხვს.
- **ლოგიკური** ანუ **ბულის** ტიპია **boolean**. ესაა სპეციალური ტიპი, რომელიც ჭეშმარიტი (**true**) ან მცდარი (**false**) მნიშვნელობის წარმოსადგენად გამოიყენება.

ეს ტიპები შეიძლება გამოყენებულ იქნას იმ სახით, როგორც არიან განსაზღვრული ან პროგრამისტის მიერ საკუთარი, ახალი ტიპების შესაქმნელად. ამრიგად, ამ ტიპების საფუძველზე შეიძლება შეიქმნას ყველა სხვა ტიპი.

პირველ ცხრილში ნაჩვენებია მთელი ტიპის მონაცემების მიერ დაკავებული მეხსიერების ზომა, დასაშვები მნიშვნელობების დიაპაზონი და სტანდარტული მნიშვნელობა გამოცხადების შემდეგ.

ცხრილი 1. **მთელი ტიპის რიცხვების თანრიგების რაოდენობა და დიაპაზონი.**

ტიპი	ბიტების რაოდენობა	დასაშვები დიაპაზონი	სტანდარტ. მნიშვნელობა
byte	8 (1 B)	-128÷127	0
short	16 (2 B)	-32768÷32767	0
int	32 (4 B)	-2147483648÷ 2147483647	0
long	64 (8 B)	-9223372036854775808 ÷ 9223372036854775807	0L

მე-2 ცხრილში ნაჩვენებია მცოცავმძიმიანი ტიპის მონაცემების მიერ დაკავებული მეხსიერების ზომა, დასაშვები მნიშვნელობების დიაპაზონი, სტანდარტული მნიშვნელობა გამოცხადების შემდეგ და სიზუსტე.

ცხრილი 2. **მცოცავმძიმიანი ტიპის რიცხვების თანრიგების რაოდენობა და დიაპაზონი**

ტიპი	ბიტების რაოდ.	დასაშვები დიაპაზონი	სტან.მნიშვნელობა	სიზუსტე
double	64 (8 B)	4.9e-324÷1.8e+308	0.0D	17
float	32 (4 B)	1.4e-045÷3.4e+038	0.0F	7-8

Java-ში სიმბოლოების შესანახად გამოიყენება **char** ტიპი.

უნდა აღვნიშნოთ, რომ ეს ტიპი ალგორითმულ ენა **C**-შიც გამოიყენება, მაგრამ ისინი ეკვივალენტური არ არის. **C**-ში **char** - ესაა 8 ბიტიანი მთელრიცხვა ტიპი. **Java**-ში კი სიმბოლოს წარმოდგენისათვის გამოიყენება **Unicode** სტანდარტი, რომელიც განსაზღვრავს სიმბოლოების საერთაშორისო ნაკრებს. ის მოიცავს ყველა ცნობილი ენების სიმბოლოებს და წარმოადგენს ათობით სხვადასხვა სიმბოლოების უნიფიცირებულ ერთობლიობას.

კონსტანტები (ლიტერალები) ასევე შეიძლება დავყოთ რიცხვით, რომელიც თავის მხრივ იყოფა მთელრიცხვა და მცოცავმძიმიან (წამდვილ) კონსტანტებად, სიმბოლურ, ლოგიკურ (ბულის) და სტრიქონულ კონსტანტებად.

ცვლადი Java პროგრამაში მონაცემთა შენახვის ძირითადი კომპონენტია. ის განისაზღვრება იდენტიფიკატორით, ტიპით და საწყისი მნიშვნელობით, რომელიც პრინციპში არასავალდებულოა. გარდა ამისა, ყველა ცვლადს აქვს განსაზღვრის არე, რომელიც განაპირობებს მის ხილვადობას სხვა ობიექტების მიმართ და არსებობის დრო.

Java-ში ცვლადების გამოცხადება მათ გამოყენებამდე უნდა მოხდეს. ცვლადების გამოცხადების ძირითადი ფორმა ასე შეიძლება წარმოვადგინოთ:

<ტიპი> <იდენტიფიკატორი> [=<მნიშვნელობა>] [,<იდენტიფიკატორი> [=მნიშვნელობა] ...];

ერთი და იგივე ტიპის რამდენიმე ცვლადის ერთდროულად გამოცხადებისათვის შეიძლება გამოვიყენოთ სია, სადაც წევრები ერთმანეთისგან გამოყოფილი იქნება მძიმით. მაგალითად:

int a, b, c; // გამოცხადებულია სამი **int** ტიპის ცვლადი:

int d = 3, e, f = 5; // სამი **int** ტიპის ცვლადი,

// ხდება **d** და **f**-ის ინიციალიზაცია

byte z = 22; // **z** ცვლადის ინიციალიზაცია

double pi = 3.14159; // **pi** ცვლადის ინიციალიზაცია

char c = `x`; // **c** ცვლადს ენიჭება `**x**`მნიშვნელობა

Java-ში შეიძლება გამოვყოთ ოპერაციების 4 ძირითადი ჯგუფი: არითმეტიკული ოპერაციები, ბიტური ოპერაციები, შედარების ოპერაციები და ლოგიკური ოპერაციები.

არითმეტიკული ოპერაციების შესრულება არ შეიძლება **boolean** ტიპის ცვლადებზე, მაგრამ შესაძლებელია **char** ტიპთან, ვინაიდან Java-ში ეს ტიპი, ფაქტიურად **int** ტიპის ქვესიმრავლეა.

მე-3 ცხრილში წარმოდგენილია არითმეტიკული ოპერაციები.

ცხრილი 3. **არითმეტიკული ოპერაციები**

ოპერაცია	**აღწერა**
+	შეკრება
-	გამოკლება
*	გამრავლება
/	გაყოფა
%	მოდულით გაყოფა
++	ინკრემენტი
+=	ავტოასოციური შეკრება
-=	ავტოასოციური გამოკლება
*=	ავტოასოციური გამრავლება
/=	ავტოასოციური გაყოფა
%=	ავტოასოციური მოდულით გაყოფა
--	დეკრემენტი

Java-ში შესაძლებელია განხორციელდეს გარკვეული ოპერაციები **long, int, short, char** და **byte** მთელრიცხვა ტიპის ბიტებზე.

მე-4 ცხრილში წარმოდგენილია ეს ოპერაციები.

ცხრილი 4. **ბიტური ოპერაციები**

ოპერაცია	აღწერა
~	ბიტური უნარული NOT (უარყოფა)
&	ბიტური AND (და)
|	ბიტური OR (ან)
^	ბიტური გამომრიცხავი OR (ან)
>>	მარჯვნივ ძვრა
>>>	მარჯვნივ ძვრა ნულებით შევსებით
<<	მარცხნივ ძვრა
&=	ავტოასოციური ბიტური AND
|=	ავტოასოციური ბიტური OR
^=	ავტოასოციური ბიტური გამომრიცხავი OR
>>=	ავტოასოციური მარჯვნივ ძვრა
>>>=	ავტოასოციური მარჯვნივ ძვრა ნულებით შევსებით
<<=	ავტოასოციური მარცხნივ ძვრა

შედარების ოპერაციებით ხდება ოპერანდების მნიშვნელობების ტოლობისა და მათი ერთმანეთის მიმართ მეტობის ან ნაკლებობის განსაზღვრა. მე-5 ცხრილში შედარების ოპერაციებია წარმოდგენილი.

ცხრილი 5. **შედარების ოპერაციები**

ოპერაცია	აღწერა
==	ტოლია
!=	არ არის ტოლი
>	მეტია
<	ნაკლებია
>=	მეტია ან ტოლი
<=	ნაკლებია ან ტოლი

მე-6 ცხრილში ბულის ლოგიკური ოპერაციებია ნაჩვენები.

ცხრილი 6. **ბულის ლოგიკური ოპერაციები**

ოპერაცია	აღწერა
&	ლოგიკური AND (და)
|	ლოგიკური OR (ან)
^	ლოგიკური XOR (გამომრიცხავი ან)
||	მოკლე OR
&&	მოკლე AND
!	ლოგიკური უნარული NOT (არა)
&=	ავტოასოციური AND (მინიჭებით)
|=	ავტოასოციური OR (მინიჭებით)
^=	ავტოასოციური XOR (მინიჭებით)
==	ტოლია
!=	არ არის ტოლი

Java-ში რეალიზებულია სპეციალური სამ ოპერანდიანი ოპერაცია, რომელმაც ზოგჯერ გარკვეულწილად **If-**

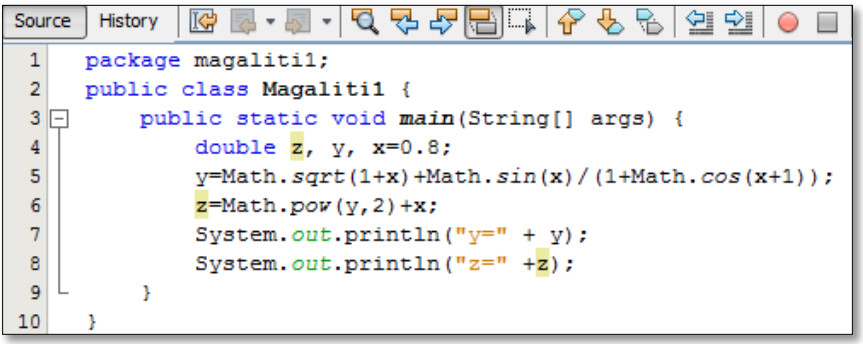
then-else ოპერატორი შეიძლება ჩაანაცვლოს. ესაა ოპერაცია „?:“, რომლის ზოგადი ფორმა შემდეგია:

<გამოსახულება1> ? <გამოსახულება2> :<გამოსახულება3>

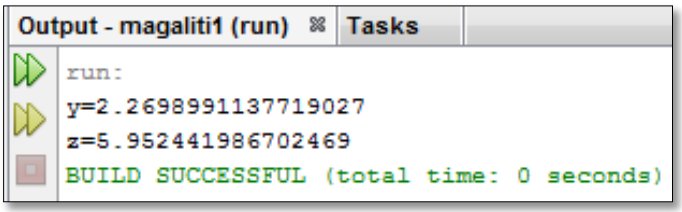
მაგალითი 1. Java-დაპროგრამების ენაზე შევადგინოთ შემდეგი გამოსახულებების გამოთვლის პროგრამა.

$$z = y^2 + x; \quad y = \sqrt{1+x} + \frac{\sin x}{1+\cos(x+1)}; \quad \text{სადაც } x=0,8.$$

პროგრამული რეალიზაცია:

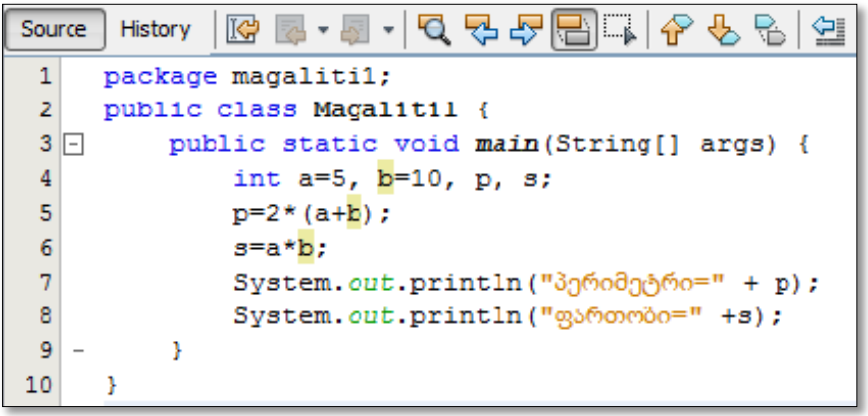


შედეგი:

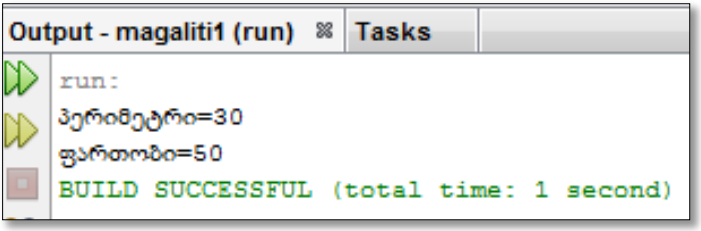


მაგალითი 2. Java-დაპროგრამების ენაზე შევადგინოთ პროგრამა, რომელიც გამოთვლის მართკუთხედის პერიმეტრსა და ფართობს, სადაც მისი არამოპირდაპირე გვერდების სიგრძეებია: a=5 სმ და b=10 სმ.

პროგრამული რეალიზაცია:



შედეგი:



დავალება

1. შეადგინეთ პროგრამა, რომელიც ჰერონის ფორმულის საფუძველზე გამოთვლის ABC სამკუთხედის ფართობს, სადაც სამკუთხედის გვერდების სიგრძეებია: a=2სმ, b=3სმ და c=4 სმ.
2. შეადგინეთ პროგრამა, რომელიც გამოთვლის კვადრატის პერიმეტრსა და ფართობს, სადაც კვადრატის გვერდის სიგრძე a=2,5 სმ-ია.
3. შეადგინეთ შემდეგი გამოსახულებების გამოთვლის პროგრამა.

   $z = |x^2 - y^2|$; $y = \frac{\sqrt{\ln A(A+B)+x}}{e^x+1}$; სადაც A=2,8 x=1,5 B=1.
4. შეადგინეთ შემდეგი გამოსახულებების გამოთვლის პროგრამა.

   $z = Ay\sin y$; $y = \sqrt{\frac{e^x+1}{e^x+2}+\sin x}$; სადაც A=10 x=1,5.
5. შეადგინეთ პროგრამა, რომელიც თქვენ მიერ შეტანილ ნებისმიერ სამნიშნა რიცხვში გამოთვლის ციფრთა ჯამს.

ლაბორატორიული სამუშაო №3

მმართველი სტრუქტურები. არჩევის ოპერატორები

არჩევის ოპერატორები საშუალებას იძლევა გამოსახულების მნიშვნელობის ან ცვლადის მდგომარეობის მიხედვით პროგრამაში არჩეულ იქნას ბრძანებების შესრულების სხვადასხვა განშტოება. Java-ში ორი არჩევის ოპერატორია რეალიზებული: **if** და **switch.**

if ოპერატორი Java პროგრამის განშტოების პირობითი შერჩევის ოპერატორია. ის გამოიყენება პროგრამის შესრულების სამართავად ორი სხვადასხვა მიმართულების შტოზე. ამ ოპერატორის ჩაწერის ზოგადი ფორმა შემდეგია:

if(<პირობა>) <ოპერატორი1>; [else <ოპერატორი2>;]

აქ თითოეული <ოპერატორი> წარმოადგენს ან რომელიმე ერთ ოპერატორს ან ერთ ბლოკში (ფიგურულ ფრჩხილებში) გაერთიანებულ რამდენიმე ოპერატორს. <პირობა> ესაა ნებისმიერი გამოსახულება, რომელიც **boolean** ტიპის შედეგს აბრუნებს. **else** აუცილებელი არაა.

if ოპერატორის მუშაობის პრინციპი შემდეგია:

თუ <პირობა> ჭეშმარიტია, მაშინ პროგრამა ასრულებს <ოპერატორი1>-ს. წინააღმდეგ შემთხვევაში იგი ასრულებს <ოპერატორი2>-ს (თუ ეს ოპერატორი არსებობს); არცერთ შემთხვევაში პროგრამა არ შეასრულებს ორივე ოპე-

რატორს. ყურადღება უნდა გავამახვილოთ, რომ **if** ან **else** საკვანძო სიტყვების შემდეგ შეიძლება მოდიოდეს, მხოლოდ ერთი ოპერატორი. თუ საჭიროა მეტი ოპერატორის მითითება, მაშინ ისინი უნდა ჩავსვათ ერთ ბლოკში (ფიგურულ ფრჩხილებში). ზოგი პროგრამისტი ფიგურულ ფრჩხილებს ერთი ოპერატორის დროსაც იყენებს. ეს აადვილებს შემდგომში ოპერატორების დამატებას. if ოპერატორის მოქმედების საილუსტრაციოდ განვიხილოთ შემდეგი

მაგალითი 1. შევადგინოთ პროგრამა, რომელიც განსაზღვრავს მაქსიმალურ მნიშვნელობას მთელი ტიპის ნებისმიერი სამი ცვლადის მნიშვნელობიდან.

პროგრამული რეალიზაცია:

```
1
2   package findmaximum;
3   public class FindMaximum {
4       public static void main(String[] args) {
5           int a=15, b=20, c=10, max;
6           max=a;
7           if(max<=b) max=b;
8           if(max<=c) max=c;
9            System.out.println("მაქსიმალური მნიშვნელობა=" + max);
10      }
11  }
```

შედეგი:

```
Output - FindMaximum (run)   Tasks
run:
მაქსიმალური მნიშვნელობა=20
BUILD SUCCESSFUL (total time: 0 seconds)
```

მაგალითი 2. შევადგინოთ პროგრამა, რომელიც დაადგენს უნაშთოდ იყოფა თუ არა რიცხვი საკუთარ ციფრთა ჯამზე.

პროგრამული რეალიზაცია:

```
package magaliti2;
public class Magaliti2 {
    public static void main(String[] args) {
        int x=999;
        int a, b, c, s, z;
        a=x/100;
        b=x/10%10;
        c=x%10;
        s=a+b+c;
        z=x/s;
        if(x%s==0){
            System.out.println("რიცხვი უნაშთოდ იყოფა თავის ციფრთა ჯამზე");
            System.out.println("შედეგი=" + z);
        }
        else
        System.out.println("რიცხვი უნაშთოდ არ იყოფა თავის ციფრთა ჯამზე");
    }
}
```

შედეგი:

```
Output - magaliti2 (run)   Tasks
run:
რიცხვი უნაშთოდ იყოფა თავის ციფრთა ჯამზე
შედეგი=37
BUILD SUCCESSFUL (total time: 1 second)
```

პროგრამებში ხშირად გვხვდება ერთმანეთში ჩალაგებული **if** ოპერატორები. ამ შემთხვევაში უნდა გვახსოვდეს, რომ **else** ოპერატორი ყოველთვის უკავშირდება მის უახლოეს **if** ოპერატორს, რომელიც იმავე ბლოკში მდებარეობს

და ჯერ არაა დაკავშირებული სხვა **else** ოპერატორთან. მაგალითად:

```
if(i == 10) {
if(j < 20) a =b;
if(k > 100) c = d; // ეს if ოპერატორი
else a = c; // დაკავშირებულია ამ else-სთან
}
else a = d; // ეს else ოპერატორი დაკავშირებულია if(i ==
                10)-თან.
```

როგორც კომენტარებიდანაც ჩანს, ბოლო **else** ოპერატორი არ არის დაკავშირებული **if(j < 20)** ოპერატორთან, ვინაიდან ის არ იმყოფება იმავე ბლოკში (მიუხედავად იმისა, რომ **if**-ის უახლოესი ოპერატორია, რომელთანაც **else** ოპერატორი ჯერ არაა დაკავშირებული). ბლოკში ბოლო **else** ოპერატორი დაკავშირებულია **if(k > 100)** ოპერატორთან, ვინაიდან იგი უახლოესია ბლოკის შიგნით.

დაპროგრამებაში გავრცელებულია კონსტრუქცია, რომელიც ჩალაგებული **if** ოპერატორების მიმდევრობისაგან აიგება, მას მრავალრგოლიან **if–else-if** სტრუქტურას უწოდებენ და შემდეგი სახე აქვს:

```
if (<პირობა>)
<ოპერატორი>;
```

else if (<პირობა>)

<ოპერატორი>;

else if (<პირობა>)

<ოპერატორი>;

else

<ოპერატორი>;

if ოპერატორი სრულდება მიმდევრობით, ზემოდან ქვემოთ. როგორც კი **if** ოპერატორის მმართავი ერთ-ერთი პირობა გახდება **true** (ჭეშმარიტი), პროგრამა შეასრულებს ამ **if** ოპერატორთან დაკავშირებულ ოპერატორს და გამოტოვებს დანარჩენ მრავალრგოლიანი სტრუქტურის ნაწილს. თუ არცერთი პირობა არ სრულდება, პროგრამა შეასრულებს ბოლო, **else** ოპერატორს. ესეიგი, თუ ყველა სხვა შემოწმების პირობა იძლევა მცდარ შედეგს, პროგრამა ასრულებს ბოლო **else** ოპერატორს. თუ ბოლო **else** ოპერატორი მითითებული არ არის და ყველა წინა შემოწმების შედეგია **false** (მცდარი), მაშინ პროგრამა ამ კონსტრუქციიდან არცერთ მოქმედებას არ შეასრულებს.

მაგალითი 3. შევადგინოთ პროგრამა, რომელიც რიცხვის ნიშანს განსაზღვრავს.

პროგრამული რეალიზაცია:

```
package magaliti3;
public class Magaliti3 {
    public static void main(String[] args) {
        int x=-5;
        if(x>0)
           System.out.println("x დადებითი რიცხვია");
        else{
            if(x<0)
               System.out.println("x უარყოფითი რიცხვია");
             else
                System.out.println("x=0");
        }
      }
    }
```

შედეგი:

```
Output - magaliti3 (run)   Tasks
run:
x უარყოფითი რიცხვია
BUILD SUCCESSFUL (total time: 0 seconds)
```

switsh ოპერატორით წარმოებს პროგრამის შესრულების განშტოება. განშტოება ხდება მმართავი გამოსახულების გამოთვლის შედეგად მიღებული მნიშვნელობის მიხედვით. **switch** ოპერატორის ზოგადი ფორმა შემდეგია:

switch (<გამოსახულება>) **{**

case <მნიშვნელობა1>:

// ოპერატორების მიმდევრობა

break;

case <მნიშვნელობა 2>:

// ოპერატორების მიმდევრობა

break;

case <მნიშვნელობა N>:

// ოპერატორების მიმდევრობა

break;

default:

// ოპერატორები

}

<გამოსახულება> უნდა იყოს **byte, short, int** ან **char** ტიპის. <მნიშვნელობა>-ში მითითებული ყოველი ტიპი <გამოსახულება>-ში მითითებული ტიპის თანადი (თავსებადი) ტიპი უნდა იყოს. (ახალი ვერსიის **JDK**-ში **switch** ოპერატორის სამართავად შესაძლებელია ჩამოთვლადი ტიპების გამოყენებაც). **case**-ს თითოეული <მნიშვნელობა> უნდა იყოს უნიკალური კონსტანტა (მუდმივა და არა ცვლადი). ასევე არაა დაშვებული **case**-ის მნიშვნელობების დუბლირება.

switch ოპერატორის მუშაობის პრინციპი შემდეგია:

<გამოსახულება>-ის მნიშვნელობა დარდება **case** ოპერატორებში მითითებულ თითოეულ კონსტანტას. თუ მოხდება რომელიმესთან თანხვედრა, მაშინ შესრულება გრძელდება ამ **case**-ის შემდეგ ჩაწერილი ოპერატორიდან. თუ არცერთი კონსტანტის მნიშვნელობა არ დაემთხვევა <გა-

მოსახულების> მნიშვნელობას, მაშინ პროგრამა ასრულებს **default** ოპერატორის შემდეგ ჩაწერილ ოპერატორებს. თუმცა, ამ ოპერატორის გამოყენება აუცილებელი არაა. **default** ოპერატორის არარსებობის და გამოსახულების არცერთ კონსტანტაზე დამთხვევის შემთხვევაში, პროგრამა **switch** კონსტრუქციაში არაფერს არ ასრულებს. პროგრამის შესრულება გრძელდება **switch** კონსტრუქციის შემდეგი ოპერატორიდან.

break ოპერატორი, რომელიც **switch** ოპერატორის შიგნითაა მითითებული, გამოიყენება ოპერატორების შესრულების მიმდევრობის წყვეტისათვის. როგორც კი პროგრამა **break** ოპერატორთან მივა, შესრულება გაგრძელდება მთლიანი **switch** ოპერატორის შემდეგი ოპერატორიდან. იგი ფაქტიურად ახორციელებს **switch** ოპერატორიდან „გამოსვლას“.

switch ოპერატორის დემონსტრირება.

მაგალითი 4. შევადგინოთ პროგრამა, რომელიც კამათლის 50-ჯერ გაგორების შემთხვევაში დათვლის თვითოეული მხარის მოსვლის ხდომილობას.

პროგრამული რეალიზაცია:

```
package kamateli;
import java.util.Random;
public class KamaTeli {
    public static void main(String[] args) {
      int x1=0, x2=0, x3=0, x4=0, x5=0, x6=0, face;
      Random rand=new Random();
      for(int i=1; i<=50; i++){
          face=(int)(1+6*rand.nextDouble());
          switch(face){
              case 1: x1++; break;
              case 2: x2++; break;
              case 3: x3++; break;
              case 4: x4++; break;
              case 5: x5++; break;
              case 6: x6++; break;
          }
      }
        System.out.println("ერთიანის მოსვლის რაოდენობა=" +x1);
         System.out.println("ორიანის მოსვლის რაოდენობა=" +x2);
          System.out.println("სამიანის მოსვლის რაოდენობა=" +x3);
           System.out.println("ოთხიანის მოსვლის რაოდენობა=" +x4);
            System.out.println("ხუთიანის მოსვლის რაოდენობა=" +x5);
             System.out.println("ექვსიანის მოსვლის რაოდენობა=" +x6);
    }
}
```

შედეგი:

```
Output - kamaTeli (run)   Tasks
run:
ერთიანის მოსვლის რაოდენობა=8
ორიანის მოსვლის რაოდენობა=13
სამიანის მოსვლის რაოდენობა=9
ოთხიანის მოსვლის რაოდენობა=6
ხუთიანის მოსვლის რაოდენობა=11
ექვსიანის მოსვლის რაოდენობა=3
BUILD SUCCESSFUL (total time: 0 seconds)
```

ცხადია, პროგრამის ყოველ შესრულებაზე გაშვებისას სხვადასხვა შედეგები მიიღება.

დავალება

1. შეადგინეთ პროგრამა, რომელიც double ტიპის ნებისმიერ სამ ცვლადს შორის განსაზღვარვს მინიმალურს.
2. შეადგინეთ პროგრამა, რომელიც გამოთვლის $y = \frac{sinx}{x}$ გამოსახულების მნიშვნელობას *x* ცვლადის ნებისმიერი მნიშვნელობის დროს (გაითვალისწინეთ, რომ 0-ზე გაყოფა დაუშვებელი ოპერაციაა!).
3. შეადგინეთ პროგრამა, რომელიც მონეტის 20-ჯერ აგდების შემთხვევაში დათვლის თვითოეული მხარის მოსვლის ხდომილობას.
4. შეადგინეთ პროგრამა, რომელიც გამოთვლის გამოსახულებათა შემდეგი სისტემის მნიშვნელობას.

$y = \frac{\sin(2-x)+1}{\sqrt{e^{A-x}}}$; სადაც *x*=2, *A*=2,8, *b*=-4,2 *c*=8, *a*=1.

$$z = \begin{cases} ax^3 - 2, & \text{თუ } y < 1; \\ ax^2 + bx + c, & \text{თუ } y = 1; \\ \text{c} & \end{cases}$$

5. შეადგინეთ პროგრამა, რომელიც მთელი ტიპის ნებისმიერ სამნიშნა რიცხვში გამოთვლის ციფრთა ნამრავლს, კონსოლზე გამოიტანს შედეგს და შეტყობინებას ლუწია თუ კენტი მიღებული შედეგი.

ლაბორატორიული სამუშაო №4

ციკლის ოპერატორები

როდესაც პროგრამული კოდის ეს თუ ის ფრაგმენტი რამდენჯერმე მეორდება, აღნიშნულ პროცესს პროგრამისტები **ციკლს** უწოდებენ. Java-ში არსებობს მარტივი ციკლის სამი სახე: **while**, **do/while** და **for**.

while ოპერატორი პროგრამებში ხშირად გამოიყენება. ის ციკლის ტანში არსებულ ოპერატორს, ან ოპერატორთა ჯგუფს იმეორებს მანამდე, სანამ ციკლის პირობა მცდარი არ გახდება. **while** ციკლის ოპერატორის ჩაწერის ფორმა შემდეგია:

```
while (<პირობა>) {
 //ციკლის ტანი;
}
```

<პირობა> შეიძლება იყოს ბულის ტიპის ნებისმიერი გამოსახულება. აღნიშნული ციკლი მუშაობს მანამ, სანამ <პირობა> ჭეშმარიტია. როგორც კი პირობა აღმოჩნდება მცდარი, პროგრამის მუშაობა გრძელდება ციკლის ტანის გარეთ არსებული მომდევნო ოპერატორიდან.

ციკლის ტანის აღმნიშვნელი ფიგურული ფრჩხილების ({ }) გამოყენება აუცილებელია, თუ მასში ერთზე მეტი ოპერატორია მოთავსებული. რადგან **while** ციკლი თავის პირო-

ბით გამოსახულებას ამოწმებს ციკლის დასაწყისში, ამიტომ, თუ გამოსახულების შედეგი მცდარია, ციკლი თავიდანვე არ შესრულდება.

მაგალითი 1. შევადგინოთ პროგრამა, რომელიც ევკლიდეს ალგორითმის საფუძველზე განსაზღვრავს ორი ნატურალური რიცხვის უდიდესი საერთო გამყოფისა და უმცირესი საერთო ჯერადის მნიშვნელობებს.

პროგრამული რეალიზაცია:

```
1   package evklide;
2   public class Evklide {
3       public static void main(String[] args) {
4           long a=240, b=360, c=a, d=b;
5           System.out.println("a=" + a);
6           System.out.println("b=" + b);
7           while(a!=b){
8               if(a>b) a-=b;
9               else
10                  b-=a;
11          }
12          System.out.println("უდიდესი საერთო გამყოფი=" + a);
13          System.out.println("უმცირესი საერთო ჯერადი=" + (c*d)/a);
14      }
15  }
```

შედეგი:

```
Output - Evklide (run)  Tasks
run:
a=240
b=360
უდიდესი საერთო გამყოფი=120
უმცირესი საერთო ჯერადი=720
```

პროგრამის წერისას, ზოგჯერ იქმენა სიტუაციები, როდესაც საჭიროა ციკლის ტანის ერთხელ მაინც შესრულება, მაშინაც კი, თუ ციკლის მმართველი გამოსახულების შედეგი მცდარია. ამ შემთხვევაში გამოიყენება **do-while** ციკლი, რომლის ჩაწერის ფორმა შემდეგია:

do {

//ციკლის ტანი;

} while(<პირობა>);

ამგვარად, **do-while** ციკლში, ჯერ მიყოლებით სრულდება ციკლის ტანში არსებული ოპერატორები და ბოლოს მომწმდება <პირობა>. ციკლი მუშაობს მანამ, სანამ პირობა ჭეშმარიტია. როგორც კი პირობა ხდება მცდარი, ციკლური პროცესი წყდება და პროგრამის შესრულება გრძელდება ციკლის ტანის გარეთ არსებული მომდევნო ოპერატორიდან. <პირობის> შედეგი, ამ შემთხვევაშიც, ბულის ტიპის უნდა იყოს.

მაგალითი 2. შევადგინოთ პროგრამა, რომელიც **y=*ax*+*b*** გამოსახულების მიხედვით წარმოგვიდგენს **y** ფუნქციისა და ***x*** არგუმენტის მნიშვნელობათა ცხრილს, სადაც ***x*∈[1;10]** ინტერვალში H=1,5-ის ტოლი ბიჯით იცვლება.

პროგრამული რეალიზაცია:

```
1   package cxrili;
2   public class Cxrili {
3       public static void main(String[] args) {
4           int a=1, b=2;
5           double x=1, y, H=1.5;
6           do{
7               y=a*x+b;
8               System.out.print("x=" + x);
9               System.out.println("\ty=" + y);
10              x+=H;
11          }while(x<=10);
12      }
13  }
```

შედეგი:

```
Output - cxrili (run)   Tasks
run:
x=1.0    y=3.0
x=2.5    y=4.5
x=4.0    y=6.0
x=5.5    y=7.5
x=7.0    y=9.0
x=8.5    y=10.5
x=10.0   y=12.0
BUILD SUCCESSFUL (total time: 1 second)
```

დაწყებული JDK5-დან, Java-ში არსებობს **for** ციკლის ორი ფორმა. პირველი - ტრადიციული ფორმა, რომელიც თავიდანვე იქნა რეალიზებული ენაში. მეორე - ახალი ფორმა **for-each**.

ტრადიცული **for** ციკლის ოპერატორის ზოგად ფორმას შემდეგი სახე აქვს:

for(<ინიციალიზაცია>; <პირობა>; <იტერაცია>) {

// ციკლის ტანი;

}

თუ ციკლში მხოლოდ ერთი ოპერატორის განმეორება ხდება, შესაძლებელია ფიგურული ფრჩხილების გამოტოვება.

for ციკლის მუშაობის პრინციპი შემდეგია: ციკლის პირველ გაშვებაზე, პროგრამა ასრულებს ციკლის ინიციალიზაციის ნაწილს. ზოგადად, ესაა გამოსახულება, რომელიც საწყის მნიშვნელობას ანიჭებს ციკლის მმართველ ცვლადს, რომელიც, ამავდროულად, ციკლის მთვლელის როლში გვევლინება. აღსანიშნავია, რომ ინიციალიზაციის გამოსახულების შესრულება ხდება მხოლოდ ერთხელ! შემდეგ პროგრამა გამოითვლის <პირობას>, რომელიც ბულის ტიპის გამოსახულება უნდა იყოს. როგორც წესი, მმართველი ცვლადი დარდება მიზნობრივ მნიშვნელობას. თუ ბულის ტიპის გამოსახულება ჭეშმარიტია, პროგრამა ასრულებს ციკლის ტანს. წინააღმდეგ შემთხვევაში, ციკლის შესრულება წყდება. ციკლის პირობის ჭეშმარიტების შემ-

თხვევაში, ციკლის ტანის შესრულების ყოველი გავლის შემდეგ სრულდება იტერაციული ნაწილი. როგორც წესი, ესაა გამოსახულება, რომელიც ზრდის ან ამცირებს ციკლის მთვლელის მნიშვნელობას. შემდეგ პროგრამა იმეორებს ციკლს: ყოველი გავლის წინ ჯერ ხდება პირობითი გამოსახულების გამოთვლა, შემდეგ სრულდება ციკლის ტანი და იტერაციული ნაწილი. პროცესი გრძელდება მანამ, სანამ ციკლის პირობა არ გახდება მცდარი.

მაგალითი 3. შევადგინოთ პროგრამა, რომელიც ითვლის შემდეგი გამოსახულების მნიშვნელობას:

$$s = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!};$$ სადაც x=1, ხოლო n=10.

პროგრამული რეალიზაცია:

```
1   package mckrivi;
2   public class Mckrivi {
3       public static void main(String[] args) {
4           double x=1, s=1, f=1, p=1, n=10;
5           for(int i=1; i<n; i++){
6               p*=x;
7               f*=i;
8               s+=p/f;
9           }
10          System.out.println("შედეგი=" + s);
11      }
12  }
```

შედეგი:

```
run:
შედეგი=2.7182815255731922
```

for ციკლი წარმოადგენს ციკლის ერთადერთ სახეს, სადაც შესაძლებელია ციკლის მმართველი პარამეტრის (მთვლელის) ინიციალიზაცია მოვახდინოთ ციკლშივე, ცხადია, თუ აღნიშნული პარამეტრი არ გამოიყენება პროგრამის სხვა ნაწილში.

მმართველი ცვლადის ციკლის ტანში გამოცხადების შემთხვევაში, უნდა გვახსოვდეს, რომ ამ ცვლადის განსაზღვრის არე და არსებობის დრო მთლიანად ემთხვევა **for** ციკლის არესა და არსებობის დროს. ანუ, ცვლადის განსაზღვრის არე შემოფარგლულია **for** ციკლით და მის გარეთ ცვლადი წყვეტს არსებობას. თუ მმართველი ცვლადი პროგრამის სხვა ადგილებშიცაა საჭირო, ის ციკლის შიგნით არ უნდა გამოცხადდეს.

თანამედროვე დაპროგრამების ენებში სულ უფრო დიდ გამოყენებას ჰპოვებს ციკლების **for-each** (თითოეულისათვის) კონცეფცია. ასეთი ციკლის დანიშნულებაა მკაცრი თანმიმდევრობით მოხდეს განმეორებადი მოქმედებები ობიექტების კოლექციის მიმართ, რომლებიც ვთქვათ, მასივადაა წარმოდგენილი.

for-each ციკლის ზოგადი ფორმა შემდეგია:

for (<ტიპი> <იტერაციული ცვლადი> : <კოლექცია>) <ოპერატორების ბლოკი>

<ტიპი> - ესაა კოლექციაში შემავალ მონაცემთა ტიპი, ხოლო <იტერაციული ცვლადი> – იტერაციული ცვლადის სახელი, რომელიც კოლექციის პირველი ელემენტიდან ბოლომდე მიმდევრობით იღებს მნიშვნელობებს.

<კოლექცია> იმ კოლექციას წარმოადგენს, რომლის მიხედვითაც ხორციელდება ციკლი.

ამრიგად, პროგრამა ციკლის თითოეულ იტერაციაზე ამოიღებს კოლექციის შემდგომ ელემენტს და ინახავს მას ცვლადში <იტერაციული ცვლადი>. ციკლი გრძელდება მანამ, სანამ არ იქნება მიღებული იტერაციის ყველა ელემენტი.

ვინაიდან იტერაციული ელემენტი კოლექციიდან იღებს მნიშვნელობებს, ამიტომ, <ტიპი> უნდა ემთხვეოდეს (ან თავსებადი იყოს) კოლექციაში არსებული ელემენტების ტიპს. საჭიროა ხაზი გავუსვათ **for-each** ციკლის ერთ მნიშვნელოვან გარემოებას. მისი იტერაციული ცვლადი წარმოადგენს ცვლადს, განკუთვნილს „მხოლოდ კითხვისთვის“. **for-each** ციკლის გამოყენების ნიმუშებს მე-5 ლაბორატორიულ სამუშაოში შემოგთავაზებთ.

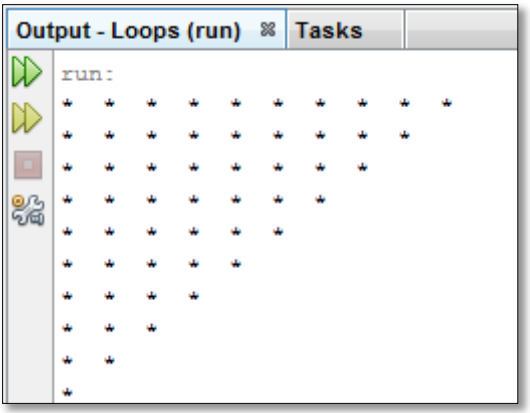
სხვა ენების ანალოგიურად, Java-ში დასაშვებია ერთმანეთში ჩალაგებული ციკლების ორგანიზება.

მაგალითი 4. შევადგინოთ პროგრამა, რომელიც ჰისტროგრამას წარმოგვიდგენს.

პროგრამული რეალიზაცია:

```
1   package loops;
2   public class Loops {
3       public static void main(String[] args) {
4           int i, j;
5           for(i=1; i<=10; i++){
6               for(j=i; j<=10; j++)
7                   System.out.print("*   ");
8               System.out.println();
9           }
10      }
11  }
```

შედეგი:



break ოპერატორი ციკლური პროცესის შეწყვეტას განაპირობებს. შესაბამისად, ციკლში არსებული ბრძანებები

აღარ სრულდება და მართვა გადაეცემა ციკლის ტანის გარეთ არსებულ მომდევნო ოპერატორს პროგრამაში.

მაგალითი 5. შევადგინოთ პროგრამა, რომელიც ციკლიდან გამოსვლას **break** ოპერატორის გამოყენებით ახორციელებს.

პროგრამული რეალიზაცია:

```
1  package loops2;
2  public class Loops2 {
3      public static void main(String[] args) {
4          for(int i=2; i<=50; i+=2){
5              if(i==20) break;
6              System.out.print(i+ "\t");
7          }
8          System.out.println();
9          System.out.println("ციკლური პროცესი შეწყვეტილია.");
10     }
11 }
```

შედეგი:

```
Output - loops2 (run)   Tasks
run:
2       4       6       8       10      12      14      16      18
ციკლური პროცესი შეწყვეტილია.
```

ზოგჯერ, კონკრეტულ იტერაციაში საჭიროა ციკლის გაგრძელება მის ტანში დანარჩენი ოპერატორების შესრულების გარეშე მოხდეს. ფაქტიურად, ესაა ციკლის ტანიდან მის ბოლოში გადასვლა. აღნიშნული მოქმედების შესასრულებლად გამოიყენება ოპერატორი **continue**. **while** და **do-while** ციკლებში **continue** ოპერატორი იწვევს მართვის გა-

დაცემას უშუალოდ ციკლის შესრულების პირობაზე. **for** ციკლში მართვა გადაეცემა ციკლის თავში იტერაციულ ნაწილს, ხოლო შემდეგ პირობით ნაწილს. სამივე ამ ციკლში ნებისმიერი შუალედური ოპერაცია გამოიტოვება.

მაგალითი 6. შევადგინოთ პროგრამა, რომელიც **continue** ოპერატორის გამოყენების დემონსტრირებას ახდენს.

პროგრამული რეალიზაცია:

```
1   package loop3;
2   public class Loop3 {
3       public static void main(String[] args) {
4           for(int i=1; i<=15; i+=2){
5               if(i==5 || i==9 || i==13)
6                   continue;
7               System.out.print(i + "\t");
8           }
9            System.out.println();
10      }
11  }
```

შედეგი:

```
Output - Loop3 (run)   Tasks
run:
1       3       7       11      15
BUILD SUCCESSFUL (total time: 0 seconds)
```

continue და **break** ოპერატორებმა შეიძლება გამოიყენონ მათი შემცველი ციკლის ჭდე.

მაგალითი 7. შევადგინოთ პროგრამა, რომელსაც 0-დან 9-მდე ციფრების გამრავლების სამკუთხა ცხრილი გამოაქვს.

პროგრამული რეალიზაცია:

```
1   package loop4;
2   public class Loop4 {
3       public static void main(String[] args) {
4           outer: for(int i=0; i<10; i++){
5               for(int j=0; j<10; j++){
6                   if(j>i){
7                       System.out.println();
8                       continue outer;
9                   }
10                  System.out.print(" " + (i*j));
11              }
12          }
13          System.out.println();
14      }
15  }
```

შედეგი:

```
Output - Loop4 (run)   Tasks
run:
 0
 0 1
 0 2 4
 0 3 6 9
 0 4 8 12 16
 0 5 10 15 20 25
 0 6 12 18 24 30 36
 0 7 14 21 28 35 42 49
 0 8 16 24 32 40 48 56 64
 0 9 18 27 36 45 54 63 72 81
```

დავალება

1. ციკლის ნებისმიერი ოპერატორის გამოყენებით შეადგინეთ პირველი 15 ნატურალური რიცხვის ჯამის, ნამრავლისა და საშუალო არითმეტიკულის გამოთვლის პროგრამა.
2. მოცემულია $y=3x^4-5x^3-x^2-6x-3$ ფუნქცია, სადაც x არგუმენტი [-2; 5,8] ინტერვალში h=0,02 ბიჯით იცვლება. შეადგინეთ პროგრამა, რომელიც განსაზღვრავს ამ ფუნქციის დადებითი მნიშვნელობების საშუალო არითმეტიკულს.
3. შეადგინეთ შემდეგი მწკრივის წევრების ჯამის გამოთვლის პროგრამა: $s = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{100}$
4. შეადგინეთ $y = \frac{cosx}{x}$ ფუნქციის მნიშვნელობათა ცხრილის გამოთვლის პროგრამა, სადაც x არგუმენტი [-10; 25] ინტერვალში h=0,1 ბიჯით იცვლება.
5. შეადგინეთ პროგრამა, რომელიც წარმოადგენს ყველა სამნიშნა რიცხვს, რომელიც უნაშთოდ იყოფა თავის ციფრთა ჯამზე.

ლაბორატორიული სამუშაო №5

მასივები

მასივი წარმოადგენს ერთი და იმავე ტიპის მონაცემთა ერთობლიობას, რომლებზეც მიმართვა საერთო სახელის გამოყენებით ხდება. Java ენაში გამოიყენება როგორც ერთ, ისე მრავალგანზომილებიანი ნებისმიერი ტიპის მასივები. მასივის კონკრეტულ ელემენტზე მიმართვა ხდება მისი ინდექსის საშუალებით. ინდექსი მთელი ტიპის მონაცემია, ან გამოსახულება, რომლის შედეგი მთელი ტიპისაა. მასივის ელემენტების რაოდენობა განსაზღვრავს მის **ზომას,** ხოლო ინდექსების რაოდენობა – **განზომილებას**.

ერთგანზომილებიანი მასივის შესაქმნელად ჯერ უნდა შეიქმნას საჭირო ტიპის მასივის ცვლადი.

ერთგანზომილებიანი მასივის გამოცხადების ზოგადი ფორმა შემდეგია:

<ტიპი> <მასივის სახელი>[ ];

აქ <ტიპი> წარმოადგენს მასივის შემადგენელი ელემენტების ტიპს. მაგალითად, შემდეგი ოპერატორით ხდება **array** მასივის გამოცხადება, რომლის ელემენტები **int** ტიპისაა:

int array[ ];

ამ გამოცხადებით ჯერ მასივი არ შექმნილა. **array** მასივის ფაქტიური მნიშვნელობა ჯერჯერობით არის **null**, რომელიც წარმოადგენს მასივს მნიშვნელობის გარეშე. **array** მასივის ცვლადის რეალური ფიზიკური მთელი რიცხვების მასივთან დასაკავშირებლად საჭიროა **new** ოპერაციით მეხსიერების გამოყოფა და მისი მისამართის ამ ცვლადზე მინიჭება.

მასივის გამოცხადების სრულყოფილი ზოგადი ფორმა შემდეგია:

<მასივის სახელი> = new <ტიპი>[<ზომა>] ;

<ტიპი> აღწერს დარეზერვებული მონაცემების მეხსიერების ტიპს. <ზომა> მიუთითებს მასივში შემავალი ელემენტების რაოდენობას, ხოლო <მასივის სახელი> მასივთან დაკავშირებული ცვლადია. ანუ, **new** ოპრაციით მეხსიერების გამოსაყოფად საჭიროა ელემენტების ტიპისა და რაოდენობის მითითება. მასივის ელემენტებისათვის **new** ოპერაციით გამოყოფილი მეხსიერების ინიციალება მოხდება ნულოვანი მნიშვნელობებით.

მაგალითად, ჩანაწერით:

array=new int [10];

ხდება 10 ელემენტიანი მასივისათვის მეხსიერების რეზერვირება და მისი დაკავშირება **array** მასივის სახელთან.

ამრიგად, მასივის შექმნა ორსაფეხურიანი პროცესია. ჯერ ცხადდება საჭირო ტიპის მასივის ცვლადი, ხოლო შემდეგ, **new** ოპერაციით გამოიყოფა მასივის შესანახი მეხსიერება და მისი მისამართი ენიჭება მასივის ცვლადს. შესაბამისად, Java-ში ყველა მასივი დინამიურად იქმნება.

მასივის გამოცხადებისა და შექმნის შემდეგ, შესაძლებელია მის თითოეულ ელემენტს მივმართოთ ინდექსით, რომელიც კვადრატულ ფრჩხილებში უნდა იყოს მოთავსებული. მასივის ინდექსაცია 0-დან იწყება.

მასივის ცვლადის გამოცხადება შეიძლება გავაერთიანოთ მისთვის მეხსიერების გამოყოფასთან:

int array[ ] = new int[10];

ასევე შესაძლებელია მასივის ინიციალიზაცია მისი გამოცხადებისას. ეს პროცესი ჰგავს ცვლადის ინიციალიზაციის პროცესს. **მასივის ინიციალიზატორი** წარმოადგენს ერთმანეთისაგან მძიმით გამოყოფილი გამოსახულებების სიას, რომელიც ფიგურულ ფრჩხილებშია მოთავსებული. მძიმით გამოყოფილია მასივის ელემენტების მნიშვნელობები. მასივი ავტომატურად იქმნება ისეთი ზომის, რომ მასში ჩაეტიოს ინიციალიზატორში მითითებული ყველა ელემენტი. ამ დროს **new** ოპერატორის გამოყენება საჭირო არ არის.

მაგალითი 1. შევადგინოთ პროგრამა, რომელიც int a[15] მასივში წრფივი ძებნის მეთოდით მოძებნის საჭირო მნიშვნელობის ელემენტებს და დათვლის მათ რაოდენობას.

პროგრამული რეალიზაცია:

```
1  package array1;
2  public class Array1 {
3      public static void main(String[] args) {
4          int a[]={10, 12, 20, 45, 57, 20, 55, 32, 43, 20, 88, 99, 20, 21, 29};
5          int searchkey=20, k=0;
6          for(int i=0; i<a.length; i++)
7              System.out.print(a[i] + "\t");
8          System.out.println();
9          for(int i=0; i<a.length; i++){
10         if(searchkey==a[i]){
11                 System.out.println("ინდექსი=" + i);
12                 k++;
13             }
14         }
15         System.out.println("საძიებელი ელემენტების რაოდენობა=" + k);
16     }
17 }
```

შედეგი:

```
run:
10      12      20      45      57      20      55      32      43      20      88      99      20      21      29
ინდექსი=2
ინდექსი=5
ინდექსი=9
ინდექსი=12
საძიებელი ელემენტების რაოდენობა=4
```

მაგალითი 2. შევადგინოთ პროგრამა, რომელიც int a[10] მასივის ელემენტებს შემთხვევითი რიცხვების გენერირების გზით მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს და „ჩაძირვის“ მეთოდით დაახარისხებს მათ კლებადობით. დაბეჭდავს საწყის და დახარისხებულ მასივებს.

პროგრამული რეალიზაცია:

```
1  package array2;
2  import java.util.Random;
3  public class Array2 {
4      public static void main(String[] args) {
5          int a[]=new int[10];
6          int temp;
7          Random rand=new Random();
8          System.out.println("საწყისი მასივი:");
9          for(int i=0; i<a.length; i++){
10             a[i]=(int)(20*rand.nextDouble());
11             System.out.print("\t" + a[i]);
12         }
13         for(int m=1; m<=a.length; m++)
14             for(int i=0; i<a.length-1; i++)
15                 if(a[i]<a[i+1]){
16                     temp=a[i];
17                     a[i]=a[i+1];
18                     a[i+1]=temp;
19                 }
20         System.out.println();
21         System.out.println("დახარისხებული მასივი:");
22         for(int i=0; i<a.length; i++)
23             System.out.print("\t" + a[i]);
24              System.out.println();
25     }
26 }
```

შედეგი:

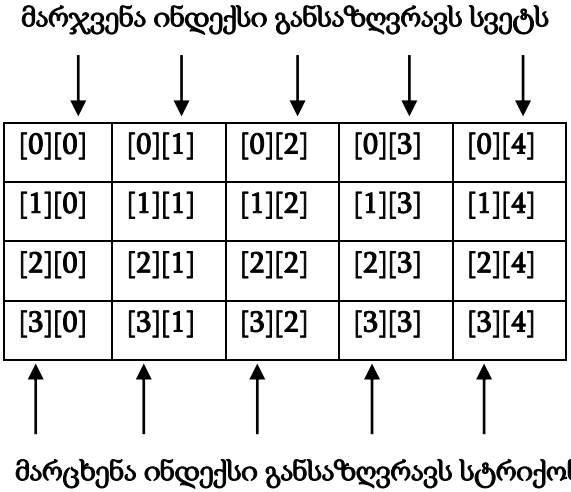
```
run:
საწყისი მასივი:
        16      7       10      2       9       1       4       2       13      7
დახარისხებული მასივი:
        16      13      10      9       7       7       4       2       2       1
```

Java-ს მრავალგანზომილებიანი მასივი წარმოადგენს მასივების მასივს. ის მოქმედებს ჩვეულებრივი მრავალ-

განზომილებიანი მასივის ანალოგიურად. თუმცა, მას გარკვეული თავისებურებაც ახასიათებთ. მრავალგანზომილებიანი მასივის თითოეული დამატებითი ინდექსის მისათითებლად იყენებენ კვადრატული ფრჩხილების ცალკე წყვილს. მაგალითად, ორგანზომილებიანი მასივი შეიძლება შემდეგი სახით გამოვაცხადოთ:

int twoD[][] = new int[4][5];

ეს ოპერატორი 4x5 განზომილებიანი მასივისთვის გამოყოფს მეხსიერებას. მასივის ლოგიკური ორგანიზება ნაჩვენებია ნახ. 1-ზე.



ნახ. 1.

შესაძლებელია მრავალგანზომილებიანი მასივების ინიციალიზაცია. ამისათვის საჭიროა თითოეული ინიციალიზატორი ჩავსვათ ცალკე ფიგურული ფრჩხილების წყვილ-

ში. მაგალითად, პროგრამის შემდეგი ფრაგმენტი ქმნის მატრიცას, სადაც ყოველი ელემენტი წარმოადგენს თავისი სტრიქონისა და სვეტის ინდექსების ნამრავლს.

```
class Matrix {
public static void main(String args[]) {
double m[] [] = {
{ 0*0, 1*0, 2*0, 3*0 },
{ 0*1, 1*1, 2*1, 3*1 },
{ 0*2, 1*2, 2*2, 3*2 },
{ 0*3, 1*3, 2*3, 3*3 }
} ;  } }
```

მასივების გამოცხადებისათვის შეიძლება გამოყენებულ იქნას მეორე ფორმა:

<ტიპი> [ ] <ცვლადის სახელი>;

ამ ფორმაში კვადრატული ფრჩხილები მიეთითება ტიპის გამოცხადების შემდეგ და არა მასივის ცვლადის შემდეგ. მაგალითად, მასივის შემდეგი ორი გამოცხადება ეკვივალენტურია:

```
int al[] = new int[3];
int [] a1= new int[3];
```

ქვემოთ ნაჩვენები ორი გამოცხადებაც ეკვივალენტურია:

```
char twod1[] [] = new char[3] [4];
char[] [] twod1 = new char[3] [4];
```

გამოცხადების ეს მეორე ფორმა მოსახერხებელია, როდესაც ხდება რამდენიმე მასივის ერთდროული გამოცხადება.

მაგალითად, გამოცხადება

int[] nums, nums2, nums3; // იქმნება 3 მასივი

ქმნის **int** ტიპის მასივების სამ სახელს. იგი ეკვივალენტურია შემდეგი გამოცხადების:

int nums[], nums2[], nums3[]; // იქმნება 3 მასივი.

მაგალითი 3. შევადგინოთ პროგრამა, რომელიც int a[5][5] კვადრატული მატრიცის ელემენტებს შემთხვევითი რიცხვების გენერირების გზით მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს, მოახდენს მატრიცის ტრანსპონირებას და დაბეჭდავს საწყის და ტრანსპონირებულ მატრიცებს.

გვინდა შევნიშნოთ, რომ კვადრატულ მატრიცაში სტრიქონებისა და სვეტების რაოდენობა ერთმანეთის ტოლია და შესაბამისად, მხოლოდ კვადრატული მატრიცის ტრანსპონირებაა შესაძლებელი.

პროგრამული რეალიზაცია:

```
4       public static void main(String[] args) {
5           int a[][]=new int[5][5];
6           int temp;
7           Random rand=new Random();
8           System.out.println("საწყისი მატრიცა");
9           for(int i=0; i<a.length; i++){
10              for(int j=0; j<a.length; j++){
11                  a[i][j]=(int)(30*rand.nextDouble());
12                  System.out.print(a[i][j] + "\t");
13              }
14              System.out.println();
15          }
16          System.out.println("ტრანსპონირებული მატრიცა");
17          for(int i=0; i<a.length; i++){
18              for(int j=i+1; j<a.length; j++){
19                  temp=a[i][j];
20                  a[i][j]=a[j][i];
21                  a[j][i]=temp;
22              }}
23          for(int i=0; i<a.length; i++){
24              for(int j=0; j<a.length; j++)
25                  System.out.print(a[i][j] + "\t");
26               System.out.println();
27              }}}
```

შედეგი:

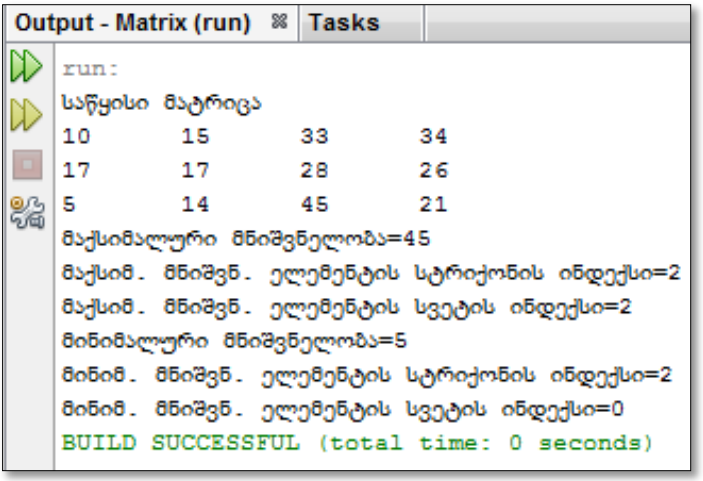
```
Output - Matrix (run)   Tasks
run:
საწყისი მატრიცა
1       21      20      13      9
6       6       22      14      19
8       29      20      16      23
29      5       26      19      17
17      28      17      28      12
ტრანსპონირებული მატრიცა
1       6       8       29      17
21      6       29      5       28
20      22      20      26      17
13      14      16      19      28
9       19      23      17      12
BUILD SUCCESSFUL (total time: 1 second)
```

მაგალითი 4. შევადგინოთ პროგრამა, რომელიც int a[3][4] კვადრატული მატრიცის ელემენტებს შემთხვევითი რიცხვების გენერირების გზით მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს, განსაზღვრავს მატრიცაში მაქსიმალური და მინიმალური მნიშვნელობის ელემენტებს და მათ ინდექსებს.

პროგრამული რეალიზაცია:

```
package matrix;
import java.util.Random;
public class Matrix {
    public static void main(String[] args) {
        int a[][]=new int[3][4];
        int maxi, mini, k1,k2, p1, p2;
        Random rand=new Random();
        System.out.println("საწყისი მატრიცა");
        for(int i=0; i<3; i++){
            for(int j=0; j<4; j++){
                a[i][j]=(int)(50*rand.nextDouble());
                System.out.print(a[i][j] + "\t");}
            System.out.println();}
        maxi=a[0][0]; mini=a[0][0];
        k1=0; p1=0; k2=0; p2=0;
        for(int i=0; i<3; i++){
            for(int j=0; j<4; j++){
                if(maxi<=a[i][j]){
                    maxi=a[i][j];
                    k1=i;
                    p1=j;}
            if(mini>=a[i][j]){
                    mini=a[i][j];
                    k2=i;
                    p2=j;}}}
          System.out.println("მაქსიმალური მნიშვნელობა=" + maxi);
          System.out.println("მაქსიმ. მნიშვნ. ელემენტის სტრიქონის ინდექსი=" + k1);
          System.out.println("მაქსიმ. მნიშვნ. ელემენტის სვეტის ინდექსი=" + p1);
          System.out.println("მინიმალური მნიშვნელობა=" +mini);
          System.out.println("მინიმ. მნიშვნ. ელემენტის სტრიქონის ინდექსი=" + k2);
          System.out.println("მინიმ. მნიშვნ. ელემენტის სვეტის ინდექსი=" + p2);}}
```

შედეგი:



მაგალითი 5. for ციკლის **for-each** სტილში გამოყენებით შევადგინოთ პროგრამა, რომელიც int b[10] მასივის ელემენტებს შემთხვევითი რიცხვების გენერირების გზით მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს, გამოთვლის მასივის ელემენტების ჯამს, ნამრავლსა და საშუალო არითმეტიკულ მნიშვნელობებს.

შევნიშნავთ, რომ ასეთი ციკლის შემთხვევაში საჭირო აღარაა მასივის ინდექსების ხელით მიწერა და ციკლის მთვლელის გამოყენება, რომლისთვისაც საწყისი და საბოლოო მნიშვნელობები უნდა მიგვეთითებინა. აქ პროგრამა ავტომატურად შეასრულებს ციკლს მასივის ყველა ელე-

მენტზე და მიმდევრობით ამოიღებს თითოეული ელემენტის მნიშვნელობას, პირველიდან ბოლო წევრამდე.

პროგრამული რეალიზაცია:

```
1   package array7;
2   import java.util.Random;
3   public class Array7 {
4       public static void main(String[] args) {
5           int b[]=new int[10];
6           float s=0;
7           double p=1;
8           Random rand=new Random();
9           System.out.println("საწყისი მასივი:");
10          for(int i=0; i<b.length; i++)
11              b[i]=(int)(15*rand.nextDouble());
12        for(int x: b){
13            System.out.print(x + "\t");
14            s+=x;
15            p*=x;
16        }
17          System.out.println();
18          System.out.println("ჯამი=" + s);
19          System.out.println("ნამრავლი=" + p);
20          System.out.println("საშუალო არითმეტიკული=" + (s/10));}}
```

შედეგი:

```
Output - Array7 (run)  Tasks
run:
საწყისი მასივი:
4       5       10      5       9       11      6       5       13      9
ჯამი=77.0
ნამრავლი=3.4749E8
საშუალო არითმეტიკული=7.7
```

დავალება

1. შეადგინეთ პროგრამა, რომელიც int b[12] მასივის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს, ცალ-ცალკე გამოთვლის მასივის კენტი და ლუწი მნიშვნელობის ელემენტების ნამრავლებს და მიღებულ შედეგებს ერთმანეთს შეადარებს.

2. შეადგინეთ პროგრამა, რომელიც int a[10] მასივის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს, კომბინირებული მეთოდით დაახარისხებს მასივის ელემენტებს ზრდადობით და დაბეჭდავს საწყის და დახარისხებულ მასივებს.

3. შეადგინეთ პროგრამა, რომელიც int c[14] მასივის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს და ბინარული ძებნის მეთოდით განსაზღვრავს საჭირო ელემენტის ინდექსს (გაითვალისწინეთ, რომ აღნიშნული მეთოდი მასივის წინასწარ დახარისხებას მოითხოვს).

4. შეადგინეთ პროგრამა, რომელიც int a[4][5] მატრიცის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს, დაახარისხებს მატრიცის ელემენტებს სტრი-

ქონების, შემდეგ სვეტების მიხედვით ზრდადობით და დაბეჭდავს საწყის და დახარისხებულ მატრიცებს.

5. შეადგინეთ პროგრამა, რომელიც int b[5][6] მატრიცის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს, გამოთვლის მატრიცის ელემენტების კვადრატების ჯამის მნიშვნელობებს სვეტების მიხედვით, განსაზღვრავს მიღებულ შედეგებს შორის უდიდესს და დაბეჭდავს: საწყის მატრიცას, მიღებულ შედეგებს და უდიდესი მნიშვნელობის შედეგის მომცემ ელემენტებს.

6. შეადგინეთ პროგრამა, რომელიც int c[4][3] მატრიცის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს, მატრიცაში სტრიქონების მიხედვით განსაზღვრავს უდიდეს ელემენტებს და მათ შორის უმცირესს. დაბეჭდავს საწყის მატრიცას და მიღებულ შედეგებს.
7. შეადგინეთ პროგრამა, რომელიც int a[10] და int b[10] ვექტორების ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს, გამოთვლის მათ სკალარულ ნამრავლს და დაბეჭდავს საწყის ვექტორებს და მიღებულ შედეგს (გაითვალისწინეთ, რომ ორი ვექტორის სკალარული ნამრავლი ამ ვექტორების შესაბამისი ელემენტების ნამრავლთა ჯამია და ის კონკრეტული რიცხვია).

ლაბორატორიული სამუშაო №6

კლასები

კლასის წევრები: ეგზემპლარის ცვლადები და მეთოდები

კლასი Java-ს ცენტრალური კომპონენტია. რადგან კლასი განსაზღვრავს ობიექტის ფორმასა და არსს, შესაბამისად, იგი წარმოადგენს ლოგიკურ კონსტრუქციას, რომლის საფუძველზეც აგებულია თავად Java დაპროგრამების ენა. ნებისმიერი კონცეფცია, რომელიც უნდა იქნას რეალიზებული Java პროგრამაში, საჭიროა განთავსდეს კლასის შიგნით.

კლასის უმნიშვნელოვანესი თვისება იმაში გამოიხატება, რომ იგი განსაზღვრავს მონაცემთა ახალ ტიპს. განსაზღვრის შემდეგ მონაცემთა ახალი ტიპი ობიექტების შესაქმნელად შეიძლება გამოვიყენოთ

ამგვარად, კლასი წარმოადგენს ობიექტის შაბლონს, ხოლო თავად ობიექტი – კლასის ეგზემპლარს. შესაბამისად, რადგან ობიექტი განიხილება, როგორც კლასის ეგზემპლარი, ამიტომ ტერმინები: **ობიექტი** და **ეგზემპლარი** ერთმანეთის სინონიმებია.

კლასის განსაზღვრა მოიცავს მისი კონკრეტული ფორმისა და არსის გამოცხადებას. ამ დროს აღიწერება კლასის მონაცემები და კოდი, რომელიც მათზე მოქმედებს. შესაძლებელია მარტივი კლასი შეიცავდეს მხოლოდ მონაცემებს ან

მხოლოდ კოდს, მაგრამ რეალურ პროგრამებში კლასების უმრავლესობა შეიცავს ორივე კომპონენტს.

კლასის გამოცხადებას ემსახურება საკვანძო სიტყვა **class**. კლასის განსაზღვრის გამარტივებულ ზოგად ფორმას შემდეგი სახე აქვს:

```
class <კლასის სახელი> {
<ტიპი> ეგზემპლარის _ცვლდი 1;
<ტიპი> ეგზემპლარის _ცვლდი 2;
// ...
<ტიპი> ეგზემპლარის _ცვლდი N;
<ტიპი> <მეთოდის სახელი 1>(<პარამეტრების სია>) {
// მეთოდის ტანი
}
<ტიპი> <მეთოდის სახელი 2>(<პარამეტრების სია>) {
// მეთოდის ტანი
}
 // ...
<ტიპი> <მეთოდის სახელი N>(<პარამეტრების სია>) {
// მეთოდის ტანი
}
}
```

კლასში განსაზღვრულ მონაცემებს ან ცვლადებს, კლასის ეგზემპლარის ცვლადებს უწოდებენ. კოდი თავსდება მეთოდის შიგნით. კლასში გამოცხადებულ მეთოდებსა და ცვლადებს ერთად **კლასის წევრები** ეწოდება. კლასების უმრავლესობაში ეგზემპლარის ცვლადებზე მოქმედებები და მიმარ-

თვები ამავე კლასში აღწერილი მეთოდების საშუალებით ხდება.

როგროც ზემოთ აღვნიშნეთ, კლასის შექმნით ჩვენ მონაცემთა ახალ ტიპს ვქმნით, რომელიც მოცემული ტიპის ობიექტების გამოსაცხადებლად შეგვიძლია გამოვიყენოთ. კლასის ობიექტების შექმნა ორსაფეხუროვან პროცესს წარმოადგენს. თავდაპირველად აუცილებელია კლასის ტიპის ცვლადის გამოცხადება. აღნიშნული ცვლადი ობიექტს არ განსაზღვრავს. ის წარმოგვიდგენს მხოლოდ ცვლადს, რომელსაც ობიექტზე მიმართვა შეუძლია. შემდეგ აუცილებელია ნამდვილი ობიექტის ფიზიკური ასლის შექმნა და მისი მინიჭება მოცემულ ცვლადზე. ეს უკანასკნელი **new** ოპერატორის საშუალებით ხორციელდება. ამ ოპერატორის ჩაწერის ზოგადი ფორმა შემდეგია:

კლასის _ ცვლადი=new კლასის_ სახელი();

აქ **კლასის_ ცვლადი** წარმოადგენს შესაქმნელი კლასის ცვლადს. **კლასის_ სახელი** კი იმ კლასის სახელია, რომლის დაკონკრეტებასაც ვახდენთ. კლასის სახელი, რომელსაც მრგვალი ფრჩხილები მოსდევს, მიუთითებს მოცემული კლასის **კონსტრუქტორზე**. ეს უკანასკნელი განსაზღვრავს მოქმედებებს, რომლებიც კლასის ობიექტის შექმნის დროს სრულდება. კონსტრუქტორები ყველა კლასის მნიშვნელოვა-

ნი ნაწილია და მათ მრავალი საინტერესო ატრიბუტი გააჩნიათ. რეალურ პროგრამებში კლასების უმრავლესობა თავიანთ კონსტრუქტორებს ცხადად განსაზღვრავს კლასის აღწერის შიგნით. ხოლო, თუ კონსტრუქტორი ცხადი სახით მითითებული არ არის, მაშინ Java კლასს ავტომატურად უზრუნველყოფს სტანდარტული კონსტრუქტორით.

როგორც წესი, კლასები ორი ელემენტისგან შედგება: ეგზემპლარის ცვლადებისა და მეთოდებისგან.

მეთოდის გამოცხადების ზოგადი ფორმა შემდეგია:

ტიპი სახელი(პარამეტრების _ სია){

//მეთოდის ტანი

}

ტიპი აქ მიუთითებს მეთოდის მიერ დასაბრუნებელი მონაცემების ტიპზე. ის შეიძლება იყოს ნებისმიერი დასაშვები ტიპი, მათ შორის პროგრამისტის მიერ შექმნილი კლასის ტიპიც. თუ მეთოდი მნიშვნელობას არ აბრუნებს, მაშინ საკვანძო სიტყვა **void** გამოიყენება.

სახელი წარმოადგენს მეთოდის სახელს. ის ნებისმიერი დასაშვები იდენტიფიკატორია, გარდა იმ სახელებისა, რომლებიც სხვა ელემენტების მიერ უკვე გამოყენებულია მიმდინარე ხედვის არეში.

პარამეტრების _ სია წარმოადგენს „ტიპი–იდენტიფიკატორი“ წყვილების მიმდევრობას, რომლებიც მძიმეებითაა გამოყოფილი. შინაარსობრივად, პარამეტრები ცვლადებია, რომლებიც იმ არგუმენტების მნიშვნელობებს იღებენ, რაც მეთოდს გადაეცემა მისი გამოძახების დროს. თუ მეთოდს პარამეტრები არ გააჩნია, მაშინ პარამეტრების სია ცარიელია.

მეთოდები, რომელთაც დაბრუნებული მნიშვნელობის ტიპად **void** არ უწერიათ, გამომძახებელ პროცედურას მნიშვნელობას **return** ოპერატორის საშუალებით უბრუნებენ. ამ უკანასკნელის ჩაწერის სინტაქსი შემდეგია:

return მნიშვნელობა;

საჭიროა გვესმოდეს, რომ, როდესაც ეგზემპლარის ცვლადზე მიმართვა სრულდება კოდით, რომელიც არ წარმოადგენს იმ კლასის შემადგენელ ნაწილს, სადაც ეგზემპლარის ცვლადია განსაზღვრული, აუცილებელია ობიექტის მითითება წერტილის ოპერატორის გამოყენებით; მაგრამ, როდესაც მიმართვა წარმოებს კოდით, რომელიც იმავე კლასის ნაწილია, რომელშიც ეგზემპლარის ცვლადია განსაზღვრული, ცვლადზე მიმართვა უშუალოდ სრულდება. ეს წესები მეთოდების მიმართაც გამოიყენება. მნიშვნელოვანია შეგვეძლოს ორი ტერმინის: **პარამეტრის** და **არგუმენტის** განსხვავება.

პარამეტრი მეთოდის მიერ განსაზღვრული ცვლადია, რომელიც მნიშვნელობას იღებს მეთოდის გამოძახების დროს. **არგუმენტი** კი მნიშვნელობაა, რომელიც მეთოდს მისი გამოძახების დროს გადაეცემა.

კლასის ყოველი ეგზემპლარის შექმნის დროს კლასის ცვლადების ინიციალება საკმაოდ დამღლელი პროცესია. გაცილებით მარტივი და მოსახერხებელია, როცა კლასის ცვლადები ობიექტის პირველი შექმნისთანავე იღებენ მნიშვნელობებს. ვინაიდან, ინიციალიზაციის მოთხოვნა ხშირია, ამიტომ Java ენა ობიექტებს უფლებას აძლევს თავად შეასრულონ ეს პროცესი მათი (ობიექტების) შექმნის დროს. ეს ავტომატური ინიციალიზაცია ხორციელდება **კონსტრუქტორის** საშუალებით.

კონსტურქტორი ობიექტის ინიციალებას უშუალოდ მისი შექმნის დროს ახდენს. მისი სახელი იმ კლასის სახელს ემთხვევა, რომელშიც ის განსაზღვრულია, ხოლო სინტაქსი – მეთოდის სინტაქსის ანალოგიურია. ობიექტის შექმნის შემდეგ კონსტრუქტორი ავტომატურად გამოიძახება **new** ოპერატორის შესრულების დამთავრებამდე. ერთი შეხედვით, კონსტრუქტორები ცოტა უცნაურად გამოიყურება, რადგან მათ არ გააჩნიათ დასაბრუნებელი შედეგის ტიპი და არც საკვანძო სიტყვა **void**-ს არ გამოიყენებენ.

მაგალითი 1. მომხმარებლის მიერ შექმნილი კლასის გამოყენებით შევადგინოთ პროგრამა, რომელიც ერთიდან 10-ის ჩათვლით ყველა ნატურალური რიცხვის ფაქტორიალებს გამოთვლის.

პროგრამული რეალიზაცია:

```
1   package factorial;
2   class student{
3       long F;
4       student(){
5           F=1;
6       }
7       void method1(){
8           for(int i=1; i<=10; i++){
9               F*=i;
10              System.out.println(i +"!=" + F);
11          }
12      }
13  }
14  public class Factorial {
15      public static void main(String[] args) {
16          student object1=new student();
17          object1.method1();
18      }
19  }
```

შედეგი:

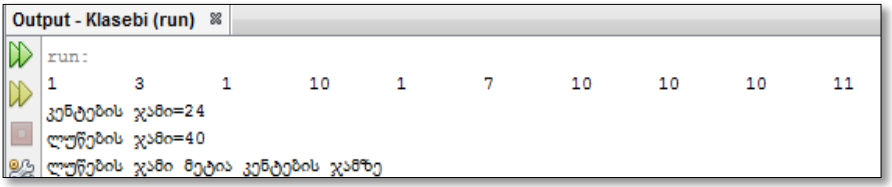
```
run:
1!=1
2!=2
3!=6
4!=24
5!=120
6!=720
7!=5040
8!=40320
9!=362880
10!=3628800
```

მაგალითი 2. მომხმარებლის მიერ შექმნილი კლასის გამოყენებით შევადგინოთ პროგრამა, რომელიც int a[10] მასივის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს. კლასის კონსტრუქტორის საშუალებით მოახდენს კლასის ეგზემპლარის ცვლადების ინიციალებას. კლასის მეთოდებით ცალ-ცალკე გამოთვლის მასივის კენტი და ლუწი მნიშვნელობის ელემენტების ჯამის მნიშვნელობებს და მიღებულ შედეგებს ერთმანეთს შეადარებს.

მინიშნება: ამოცანის გადაწყვეტის დროს გათვალისწინებულ იქნა ის გარემოება, რომ ნებისმიერი ლუწი მთელი რიცხვის 2-ზე გაყოფის შედეგად ნაშთი ნოლის ტოლი მიიღება, ხოლო კენტი რიცხვის შემთხვევაში - ერთი.

ქვემოთ წარმოდგენილია დასმული ამოცანის შედეგები და შემდგომ ამოცანის პროგრამული რეალიზაცია.

შედეგები:



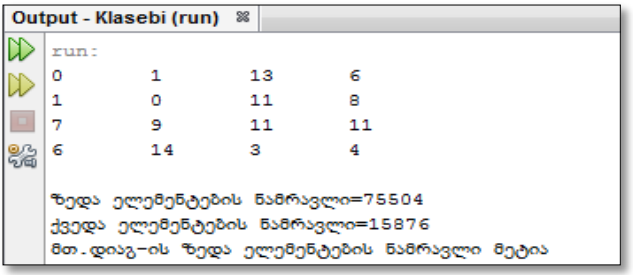
პროგრამული რეალიზაცია:

```
1   package klasebi;
2   import java.util.Random;
3   class student{
4       int a[]=new int[10];
5       int s1, s2;
6       student(){
7           s1=0; s2=0;
8           Random rand=new Random();
9           for(int i=0; i<a.length; i++){
10              a[i]=(int)(12*rand.nextDouble());
11              System.out.print(a[i]+ "\t");}}
12      int method1(){
13        for(int i=0; i<a.length; i++)
14           if(a[i]%2==1)
15               s1+=a[i];
16        return s1;}
17    int method2(){
18        for(int i=0; i<a.length; i++)
19           if(a[i]%2==0)
20               s2+=a[i];
21        return s2;}
22    void method3(int s1, int s2){
23        if(s1>s2)
24            System.out.println("კენტების ჯამი მეტია ლუწების ჯამზე ");
25        else{
26            if(s1<s2)
27                System.out.println("ლუწების ჯამი მეტია კენტების ჯამზე");
28            else
29                System.out.println("შედეგები ერთმანეთის ტოლია");}}}
30        public class Klasebi {
31      public static void main(String[] args) {
32          student object1=new student();
33          int res1, res2;
34          res1=object1.method1();
35          res2=object1.method2();
36          System.out.println();
37          System.out.println("კენტების ჯამი=" + res1);
38          System.out.println("ლუწების ჯამი=" + res2);
39          object1.method3(res1, res2);}}
```

მაგალითი 3. მომხმარებლის მიერ შექმნილი კლასის გამოყენებით შევადგინოთ პროგრამა, რომელიც int a[4][4] მატრიცის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს. კლასის კონსტრუქტორის საშუალებით მოახდენს კლასის ეგზემპლარის ცვლადების ინიციალებას. კლასის მეთოდებით ცალ-ცალკე გამოთვლის მატრიცის მთავარი დიაგონალის ზემოთ და ქვემოთ არსებული ელემენტების ნამრავლებს და მიღებულ შედეგებს ერთმანეთს შეადარებს.

მინიშნება: ამოცანის გადაწყვეტის დროს გათვალისწინებულ იქნა ის გარემოება, რომ კვადრატული მატრიცის მთავარი დიაგონალის ზემოთ არსებული ელემენტების სტრიქონების ინდექსების მნიშვნელობები ნაკლებია ამავე ელემენტების სვეტების ინდექსების მნიშვნელობებზე, ხოლო მთავარი დიაგონლის ქვემოთ განლაგებული ელემენტების შემთხვევაში კი პირიქით - სტრიქონების ინდექსების მნიშვნელობები მეტია სვეტების ინდექსების მნიშვნელობებზე.

შედეგები:



პროგრამული რეალიზაცია:

```
package klasebi;
import java.util.Random;
class student{
    int a[][]=new int[4][4];
    long s1, s2;
    student(){
        s1=1; s2=1;
        Random rand=new Random();
        for(int i=0; i<a.length; i++){
            for(int j=0; j<a.length; j++){
            a[i][j]=(int)(15*rand.nextDouble());
            System.out.print(a[i][j]+ "\t");}
            System.out.println();}}
    long method1(){
      for(int i=0; i<a.length; i++)
        for(int j=0; j<a.length; j++)
         if(i<j) s1*=a[i][j];
      return s1;}
  long method2(){
      for(int i=0; i<a.length; i++)
        for(int j=0; j<a.length; j++)
         if(i>j) s2*=a[i][j];
      return s2;}
  void method3(long s1, long s2){
      if(s1>s2)
          System.out.println("მთ.დიაგ-ის ზედა ელემენტების ნამრავლი მეტია");
      else{
          if(s1<s2)
              System.out.println("მთ.დიაგ-ის ქვედა ელემენტების ნამრავლი მეტია");
          else
              System.out.println("შედეგები ერთმანეთის ტოლია");}}}
      public class Klasebi {
    public static void main(String[] args) {
        student object1=new student();
        long res1, res2;
        res1=object1.method1();
        res2=object1.method2();
        System.out.println();
        System.out.println("ზედა ელემენტების ნამრავლი=" + res1);
        System.out.println("ქვედა ელემენტების ნამრავლი=" + res2);
        object1.method3(res1, res2);}}
```

ზოგჯერ საჭიროა, რომ მეთოდმა მის გამომძახებელ ობიექტს მიმართოს. ამ მიზნით Java–ში განსაზღვრულია **this** საკვანძო სიტყვა. ის შეიძლება ნებისმიერი მეთოდის შიგნით მიმდინარე ობიექტზე მიმართვისათვის გამოვიყენოთ.

ასევე ცნობილია, რომ Java ენაში ერთი და იგივე ხედვის არეში დაუშვებელია ერთი და იმავე სახელის ორი ლოკალური ცვლადის გამოცხადება. საჭიროა შევნიშნოთ, რომ შესაძლებელია ლოკალური ცვლადების, მათ შორის მეთოდის ფორმალური პარამეტრების არსებობა, რომელთა სახელებიც ემთხვევა კლასის ეგზემპლარის ცვლადების სახელებს. მაგრამ, როდესაც ლოკალური ცვლადის სახელი ემთხვევა ეგზემპლარის ცვლადის სახელს, მაშინ ლოკალური ცვლადი ფარავს ეგზემპლარის ცვლადს. რადგან **this** საკვანძო სიტყვა საშუალებას გვაძლევს უშუალოდ მივმართოთ ობიექტს, ამიტომ მისი გამოყენება შეგვიძლია სახელების სივრცის ნებისმიერი კონფლიქტის გადასაწყვეტად, მათ შორის ისეთი კონფლიქტების, რომელიც ეგზემპლარის ცვლადებს და ლოკალურ ცვლადებს უკავშირდება.

ზემოთ თქმულის საილუსტრაციოდ განვიხილოთ

მაგალითი 4. მომხმარებლის მიერ შექმნილი კლასის და **this** საკვანძო სიტყვის გამოყენებით შევადგინოთ მართკუთ-

ხა პარალელეპიპედის მოცულობის გამოსათვლელი პროგ-რამა.

პროგრამული რეალიზაცია:

```
1   package box;
2   class student{
3       int a, b, c;
4       student(int a, int b, int c){
5           this.a=a;
6           this.b=b;
7           this.c=c;
8           System.out.println("a=" + a);
9           System.out.println("b=" + b);
10          System.out.println("c=" + c);
11      }
12      int method1(){
13          return a*b*c;
14      }
15  }
16  public class Box {
17      public static void main(String[] args) {
18          System.out.println("კლასის კონსტრუქტორი:");
19         student object1=new student(2, 3, 4);
20          System.out.println("მოცულობა=" + object1.method1()); }}
```

შედეგი:

```
Output - Box (run)
run:
კლასის კონსტრუქტორი:
a=2
b=3
c=4
მოცულობა=24
```

დავალება

1. მომხმარებლის მიერ შექმნილი კლასის გამოყენებით შეადგინეთ პროგრამა, რომელიც int b[15] მასივის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს. კლასის კონსტრუქტორის საშუალებით მოახდენს კლასის ეგზემპლარის ცვლადების ინიციალებას. კლასის მეთოდებით ცალ-ცალკე გამოთვლის მასივის კენტი და ლუწინდექსიანი ელემენტების საშუალო არითმეტიკულ მნიშვნელობებს და მიღებულ შედეგებს ერთმანეთს შეადარებს.

2. მომხმარებლის მიერ შექმნილი კლასის გამოყენებით შეადგინეთ პროგრამა, რომელიც int a[5][5] მატრიცის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს. კლასის კონსტრუქტორის საშუალებით მოახდენს კლასის ეგზემპლარის ცვლადების ინიციალებას. კლასის მეთოდებით ცალ-ცალკე გამოთვლის მატრიცის არამთავარი დიაგონალის ზემოთ და ქვემოთ არსებული ელემენტების კვადრატების ჯამის მნიშვნელობებს და მიღებულ შედეგებს ერთმანეთს შეადარებს.

3. მომხმარებლის მიერ შექმნილი კლასის და **this** საკვანძო სიტყვის გამოყენებით შეადგინეთ სამკუთხედის ფართო-

ბის გამოსათვლელი პროგრამა (დაიცავით სამკუთხედების აგების წესი).

4. მომხმარებლის მიერ შექმნილი კლასის გამოყენებით შეადგინეთ პროგრამა, რომელიც int D[13] მასივის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს. კლასის კონსტრუქტორის საშუალებით მოახდენს კლასის ეგზემპლარის ცვლადების ინიციალებას. კლასის მეთოდების საშუალებით დაახარისხებს მასივის ელემენტებს კლებადობით „მარტივი გადანაცვლების“ მეთოდით და დაბეჭდავს საწყის და დახარისხებულ მასივებს.

5. მომხმარებლის მიერ შექმნილი კლასის გამოყენებით შეადგინეთ პროგრამა, რომელიც int a[3][5] მატრიცის ელემენტებს მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს. კლასის კონსტრუქტორის საშუალებით მოახდენს კლასის ეგზემპლარის ცვლადების ინიციალებას. კლასის მეთოდებით გამოთვლის მატრიცის ელემენტების ნამრავლებს სვეტების მიხედვით და მიღებულ შედეგებს შორის განსაზღვრავს უმცირესს.

ლაბორატორიული სამუშაო №7

მეთოდების გადატვირთვა

დაპროგრამების ენა Java უფლებას გვაძლევს ერთსა და იმავე კლასში განვსაზღვროთ ერთი და იგივე სახელის მქონე ორი ან მეტი მეთოდი, მაგრამ მხოლოდ იმ შემთხვევაში, თუ მეთოდებს განსხვავებული პარამეტრები გააჩნიათ. აღნიშნულ შემთხვევაში მეთოდებს **გადატვირთულს,** ხოლო თავად პროცესს კი – **მეთოდების გადატვირთვას** უწოდებენ.

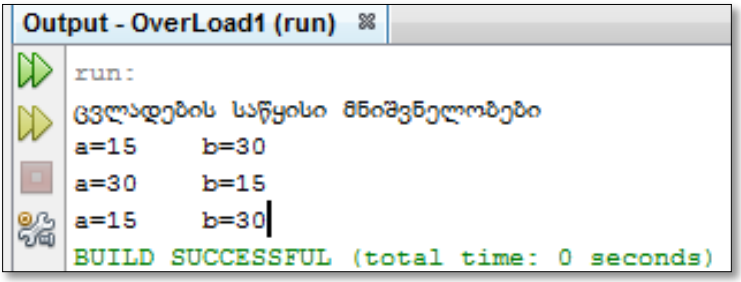
გადატვირთული მეთოდის გამოძახების დროს საჭირო ვერსიის განსაზღვრისათვის Java იყენებს მეთოდის პარამეტრების ტიპს და/ან მათ რაოდენობას. შესაბამისად, გადატვირთული მეთოდები ერთმანეთისგან პარამეტრების ტიპებით და /ან მათი რაოდენობით უნდა განსხვავდებოდეს. მართალია, გადატვირთულ მეთოდებს შესაძლოა განსხვავებული დასაბრუნებელი შედეგის ტიპი ქონდეთ, მაგრამ დასაბრუნებელი შედეგის ტიპი არ არის საკმარისი მეთოდის ორი ან მეტი განსხვავებული ვერსიის განსაზღვრისთვის. გადატვირთული მეთოდის გამოძახების დროს Java-ს შესრულების გარემო ამუშავებს მეთოდის იმ ვერსიას, რომლის პარამეტრები გამოძახებული მეთოდის არგუმენტებს შეესაბამება.

გადატვირთული მეთოდის დემონსტრირების მიზნით განვიხილოთ შემდეგი **მაგალითი 1.** შევადგინოთ პროგრამა, რომელიც მეთოდის გადატვირთვის გზით ორ ცვლადს მნიშვნელობებს უცვლის (ერთმანეთის მნიშვნელობებს ანიჭებს).

პროგრამული რეალიზაცია:

```
1   package overload1;
2   class student{
3       int a, b;
4       student(int a, int b){
5           System.out.println("ცვლადების საწყისი მნიშვნელობები");
6           this.a=a;
7           this.b=b;
8       }
9       void method1(){
10          System.out.println("a=" +a + "\tb=" + b);
11
12      }
13      void method2(int a, int b){
14          int temp=this.a;
15          this.a=this.b;
16          this.b=temp;
17      }
18     void method2(){
19          a+=b;
20          b=a-b;
21          a=a-b;
22      }
23  }
24  public class OverLoad1 {
25      public static void main(String[] args) {
26          student object1=new student(15, 30);
27          object1.method1();
28          object1.method2(15, 30);
29          object1.method1();
30          object1.method2();
31          object1.method1();
32      }
33  }
```

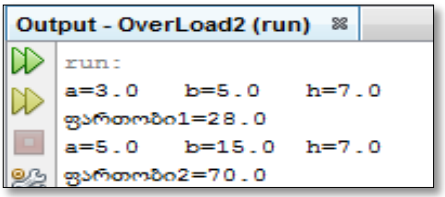
შედეგი:



აღსანიშნავია, რომ პარამეტრებისა და არგუმენტების სრული შესაბამისობა ყოველთვის სავალდებულო არ არის. ზოგ შემთხვევაში, გადატვირთული მეთოდის ვერსიის მოძებნის დროს ტიპების ავტომატური გარდასახვა გამოიყენება.

სხვა მეთოდების მსგავსად, შესაძლებელია კონსტრუქტორების გადატვირთვაც. აღნიშნულის საილუსტრაციოდ განვიხილოთ **მაგალითი 2.** შევადგინოთ პროგრამა, რომელიც კლასის კონსტრუქტორის გადატვირთვის გამოყენებით გამოთვლის ტრაპეციის ფართობს. ქვემოთ წარმოდგენილია დასმული ამოცანის ამოხსნის შედეგი და შემდგომ მისი პროგრამული რეალიზაცია.

შედეგი:



პროგრამული რეალიზაცია:

```
Source | History
1  package overload2;
2  class student{
3      double a, b, h;
4      student(double a, double b, double h){
5          this.a=a;
6          this.b=b;
7          this.h=h;
8      }
9      student(){
10         a=5;
11         b=a+10;
12         h=7;
13     }
14     void method1(){
15         System.out.println("a=" + a + "\tb=" +b + "\th=" +h);
16     }
17     double method2(){
18         return ((a+b)/2*h);
19     }
20 }
21 public class OverLoad2 {
22     public static void main(String[] args) {
23         student object1=new student(3, 5, 7);
24         student object2=new student();
25         object1.method1();
26         System.out.println("ფართობი1=" + object1.method2());
27         object2.method1();
28     System.out.println("ფართობი2=" + object2.method2());
29     }
30 }
```

მოცემულ მაგალითში, პირველ შემთხვევაში ეგზემპლარის ცვლადებზე მნიშვნელობების მისანიჭებლად კონსტრუქტორის მიერ გამოყენებულ იქნა **this** საკვანძო სიტყვა, ხოლო მეორე შემთხვევაში - გარკვეული ფორმულები.

დავალება

1. შეადგინეთ პროგრამა, რომელიც მეთოდის გადატვირთვის გზით მთელი ტიპის ნებისმიერ სამ ცვლადს შორის განსაზღვრავს შუალედური მნიშვნელობის ცვლადს.
2. შეადგინეთ პროგრამა, რომელიც მეთოდის გადატვირთვის გზით, შემდეგი ფორმულის $S = 2\pi RH$ საფუძველზე, გამოთვლის ცილინდრის გვერდითი ზედაპირის ფართობს.
3. შეადგინეთ პროგრამა, რომელიც კლასის კონსტრუქტორის გადატვირთვის გზით, $S = \pi Rl$ ფორმულის საფუძველზე, გამოთვლის კონუსის გვერდითი ზედაპირის ფართობს.
4. შეადგინეთ პროგრამა, რომელიც კლასის კონსტრუქტორის გადატვირთვის გზით მთელი ტიპის ნებისმიერ ოთხნიშნა რიცხვში განსაზღვრავს საკუთარი ციფრების კუბების ჯამს.
5. შეადგინეთ პროგრამა, რომელიც კლასის კონსტრუქტორისა და მეთოდის გადატვირთვის გზით გამოთვლის მთელი ტიპის ნებისმიერ სამ ცვლადს შორის არსებული უდიდესი და უმცირესი მნიშვნელობების მქონე ცვლადების ნახევარ-ჯამს.

ლაბორატორიული სამუშაო №8

ობიექტების გამოყენება პარამეტრების როლში. არგუმენტების მეთოდებში გადაცემის გზები. მეთოდიდან ობიექტების დაბრუნება

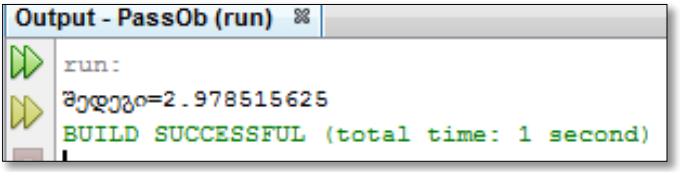
აქამდე მეთოდების პარამეტრების როლში ჩვენ მხოლოდ მონაცემთა მარტივ ტიპებს ვიყენებდით. თუმცა, Java-ში შესაძლებელია მეთოდებს პარამეტრების სახით ობიექტებიც გადავცეთ.

ზემოაღნიშნულის საილუსტრაციოდ განვიხილოთ შემდეგი **მაგალითი 1.** შევადგინოთ შემდეგი მწკრივის წევრების ჯამის გამოთვლის პროგრამა $s = 1 + \frac{1}{x} + \frac{2}{x^2} + \frac{3}{x^3} + \cdots + \frac{n}{x^n}$.

პროგრამული რეალიზაცია:

```
1   package passob;
2   class student{
3       double s, p, n, x;
4       student(){
5           s=1; p=1; n=10; x=2;
6       }
7       double method1(student k){
8           for(int i=1; i<n; i++){
9             k.p=(k.p)*x;
10            k.s=(k.s)+i/(k.p);
11          }
12            return (k.s);
13          }}
14      public class PassOb {
15      public static void main(String[] args) {
16          student object1=new student();
17          student object2=new student();
18          System.out.println("შედეგი=" + object2.method1(object1));
19      }}
```

შედეგი:

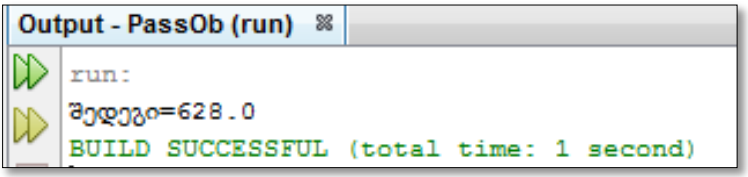


ზემოთ წარმოდგენილ პროგრამაში method1()–ში პარამეტრის როლში გადაცემულია student კლასის ობიექტი, რომელსაც თავისუფალი წვდომა აქვს ამავე კლასის მონაცემებზე.

ხშირ შემთხვევაში კონსტრუქტორებში პარამეტრების სახით ობიექტები გადაიცემა. აღნიშნულის საილუსტრაციოდ განვიხილოთ **მაგალითი 2.** შევადგინოთ პროგრამა, რომელიც ცილინდრის მოცულობას შემდეგი ფორმულის $V = \pi R^2 H$ საფუძველზე გამოთვლის.

ამოცანის გადაწყვეტის შედეგი ქვემოთ არის წარმოდგენილი. შემდეგ კი ნაჩვენებია მისი პროგრამული რეალიზაცია.

შედეგი:



პროგრამული რეალიზაცია:

```
1   package passob;
2   class student{
3       double pi, R, H;
4       student(double pi, double R, double H){
5         this.pi=pi;
6         this.R=R;
7         this.H=H;
8       }
9       student(student ob){
10          pi=ob.pi;
11          R=ob.R;
12          H=ob.H;
13      }
14      double method1(){
15          return (pi*R*R*H);
16      }
17  }
18      public class PassOb {
19      public static void main(String[] args) {
20          student object1=new student(3.14, 5, 8);
21          student object2=new student(object1);
22          System.out.println("შედეგი=" + object2.method1());}}
```

ამგვარად, შეგვიძლია დავასკვნათ, რომ როდესაც საკუთარი კლასების შემუშავებას შევუდგებით, ობიექტების ეფექტურად შექმნისათვის სასურველია კონსტრუქტორების სხვადასხვა ფორმით ვისარგებლოთ.

ზოგად შემთხვევაში, არსებობს ორი საშუალება, რომელთა გამოყენებით კომპიუტერულ ენას შეუძლია ქვეპროგრამას გადასცეს არგუმენტები. პირველი გზაა – **გამოძახება მნიშვნელობით.** ამ მიდგომის გამოყენების შემთხვევაში, ადგილი აქვს არგუმენტის მნიშვნელობის კოპირებას ქვეპროგ-

რამის ფორმალურ პარამეტრში. შესაბამისად, ქვეპროგრამის პარამეტრზე შესრულებული ცვლილება გავლენას ვერ ახდენს არგუმენტზე. არგუმენტის გადაცემის მეორე გზაა – **დამისამართებით გამოძახება.** ამ მიდგომის გამოყენებისას, პარამეტრს არგუმენტზე მიმართვა გადაეცემა (და არა მისი მნიშვნელობა). ქვეპროგრამის შიგნით ეს გამოიყენება იმ რეალურ არგუმენტზე მიმართვისათვის, რომელიც გამოძახების ბრძანებაშია მითითებული. ეს ნიშნავს, რომ ქვეპროგრამის გამოძახების დროს პარამეტრზე განხორციელებული ცვლილებები გავლენას ახდენს შესაბამის არგუმენტზე.

როგორც წესი, Java–ში ყველა არგუმენტის გადაცემა მნიშვნელობით სრულდება, მაგრამ კონკრეტული ეფექტი დამოკიდებულია იმაზე, თუ რის გადაცემას ვახდენთ: საბაზო ტიპის, თუ მისამართის. როდესაც მეთოდს ელემენტარული ტიპი გადაეცემა, მაშინ აღნიშნული პროცესი მნიშვნელობით ხორციელდება. ამგვარად, იქმნება არგუმენტის ასლი და პარამეტრზე შესრულებული ცვლილებები გავლენას ვერ ახდენს შესაბამის არგუმენტზე მეთოდის გარეთ.

ზემო აღნიშნულის საილუსტრაციოდ განვიხილოთ **მაგალითი 3.** შევადგინოთ პროგრამა, სადაც თვალსაჩინოა მეთოდის დამისამართებით და მნიშვნელობით გამოძახებულ არგუმენტებს შორის სხვაობა. პროგრამა კვადრატული

ფესვის მნიშვნელობას იღებს კონკრეტული ცვლადიდან და მიღებულ შედეგს იმავე ცვლადს ანიჭებს.

პროგრამული რეალიზაცია

```
1  package parametrs1;
2  class student{
3      double x, y;
4      student(double k, double p){
5          x=k;
6          y=p;
7          System.out.println("x=" + x);
8          System.out.println("y=" + y);
9      }
10     void method1(){
11         x=Math.sqrt(x);
12         System.out.println("შედეგი-1=" + x);
13     }
14     void method2(student ob){
15         ob.y=Math.sqrt(ob.y);
16         System.out.println("შედეგი-2=" + ob.y);
17     }
18 }
19 public class Parametrs1 {
20     public static void main(String[] args) {
21         System.out.println("ცვლადების საწყისი მნიშვნელობები:");
22         double x=625, y=81;
23         student object1=new student(x, y);
24         object1.method1();
25      System.out.println("x-ის მნიშვნელობა მეთოდის მუშობის შემდეგ=" + x);
26         object1.method2(object1);
27      System.out.println("y-ის მნიშვნელობა მეთოდის მუშობის შემდეგ=" + object1.y);
28
29     }}
```

შედეგი:

```
run:
ცვლადების საწყისი მნიშვნელობები:
x=625.0
y=81.0
შედეგი-1=25.0
x-ის მნიშვნელობა მეთოდის მუშობის შემდეგ=625.0
შედეგი-2=9.0
y-ის მნიშვნელობა მეთოდის მუშობის შემდეგ=9.0
```

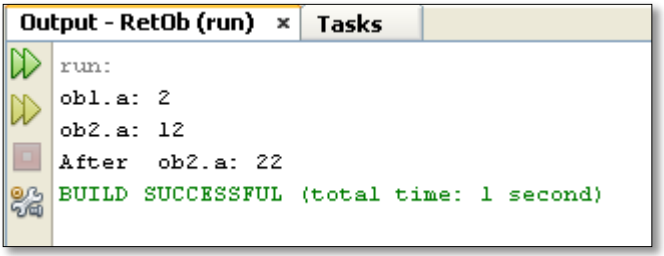
მეთოდს შეუძლია მონაცემთა ნებისმიერი ტიპის, მათ შორის მომხმარებლის მიერ შექმნილი კლასების ტიპების დაბრუნებაც. აღნიშნულის საილუსტრაციოდ განვიხილოთ **მაგალითი 4.** შევადგინოთ პროგრამა, სადაც incrByTen() სახელის მქონე მეთოდი აბრუნებს ობიექტს, რომელშიც *a* ცვლადის მნიშვნელობა 10–ით მეტია გამომძახებელი ობიექტის შესაბამისი ცვლადის მნიშვნელობაზე.

პროგრამული რეალიზაცია

Source History

```
1
2   package retob;
3   class Test{
4       int a;
5       Test(int i){
6           a=i;
7       }
8       Test incrByTen(){
9           Test temp=new Test(a+10);
10          return temp;
11      }
12  }
13  public class RetOb {
14      public static void main(String[] args) {
15          Test ob1=new Test(2);
16          Test ob2;
17          ob2=ob1.incrByTen();
18          System.out.println("ob1.a: " + ob1.a);
19          System.out.println("ob2.a: " + ob2.a);
20           ob2=ob2.incrByTen();
21           System.out.println("After  ob2.a: " + ob2.a);
22      }
23  }
24
```

შედეგი:



<u>დავალება</u>

1. შეადგინეთ პროგრამა, რომელიც კონკრეტული ცვლადის მნიშვნელობას აიყვანს კუბში და მიღებულ შედეგს იმავე ცვლადს მიანიჭებს. ამოცანის გადასაწყვეტად გამოიყენეთ ორი მეთოდი, ამათგან პირველში არგუმენტი გამოიძახეთ მნიშვნელობით, ხოლო მეორეში - დამისამართებით. გააკეთეთ სათანადო დასკვნები.

2. შეადგინეთ პროგრამა, რომელიც int a[12] მასივის ელემენტებს [1;10] ინტერვალიდან მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს და გამოთვლის მასივის 7-ზე ნაკლები მნიშვნელობის მქონე ელემენტების საშუალო არითმეტიკულს. ამოცანის გადასაწყვეტად გამოიყენეთ მეთოდი, რომელშიც პარამეტრის სახით ობიექტია გადაცემული.

3. შეადგინეთ პროგრამა, რომელიც მართკუთხა სამკუთხედში გამოთვლის ჰიპოტენუზის სიგრძეს კათეტების მოცემული მნიშვნელობების დროს. ამოცანის გადასაწყვეტად გამოიყენეთ მეთოდი, რომელშიც პარამეტრის სახით ობიექტია გადაცემული.

4. შეადგინეთ პროგრამა, რომელიც int b[15] მასივის ელემენტებს [1;25] ინტერვალიდან მიანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს და გამოთვლის მასივის 10-ზე მეტი მნიშვნელობის მქონე ელემენტების კვადრატების ჯამს. ამოცანის გადასაწყვეტად გამოიყენეთ მეთოდი, რომელშიც პარამეტრის სახით ობიექტია გადაცემული.

5. შეადგინეთ პროგრამა, რომელიც მე-5 ხარისხის ფესვს ამოიღებს კონკრეტული ცვლადიდან და მიღებულ შედეგს იმავე ცვლადს მიანიჭებს. ამოცანის გადასაწყვეტად გამოიყენეთ ორი მეთოდი, ამათგან პირველში არგუმენტი გამოიძახეთ მნიშვნელობით, ხოლო მეორეში - დამისამართებით. გააკეთეთ სათანადო დასკვნები.

ლაბორატორიული სამუშაო №9

რეკურსია

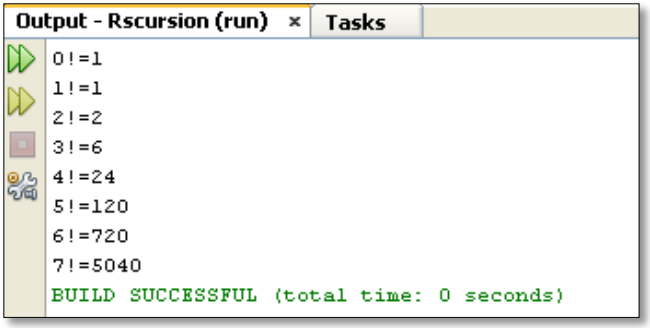
რეკურსია არის ატრიბუტი, რომელიც მეთოდს საკუთარი თავის გამოძახების უფლებას აძლევს. ასეთ დროს, თავად მეთოდს **რეკურსიული მეთოდი** ეწოდება.

რეკურსიის კლასიკურ მაგალითს არაუარყოფითი მთელი რიცხვის ფაქტორიალის გამოთვლა წარმოადგენს. N რიცხვის ფაქტორიალი ერთიდან N-მდე მთელი რიცხვების ნამრავლია. მაგალითად 5–ის ფაქტორიალი უდრის 1*2*3*4*5 ანუ 120–ს.

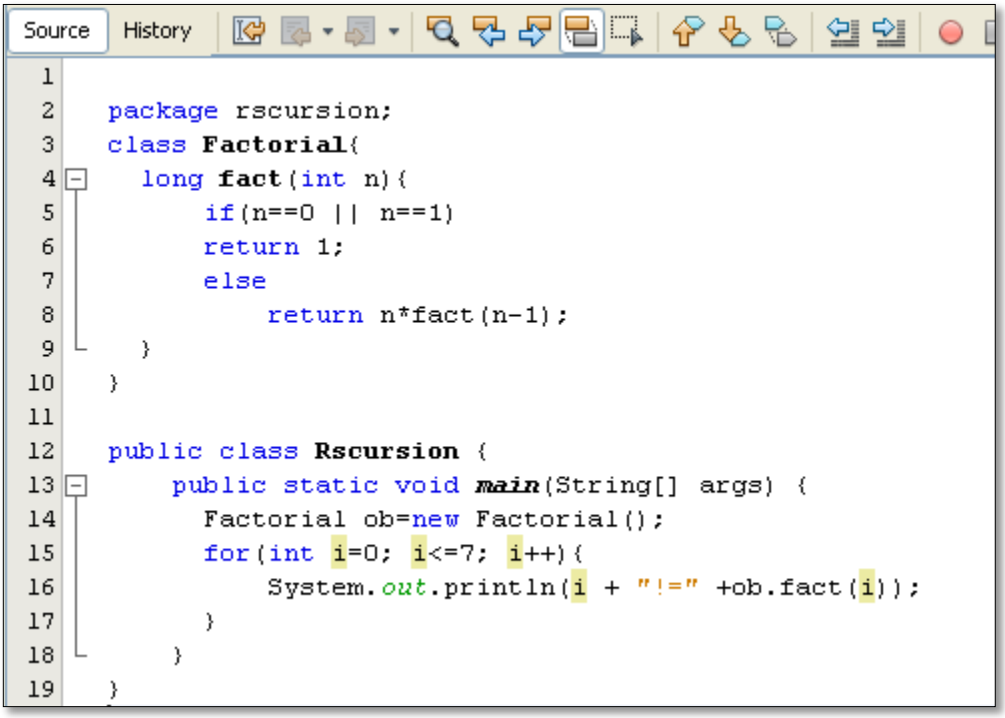
ახლა კი განვიხილოთ **მაგალითი 1.** შევადგინოთ პროგრამა, რომელიც რეკურსიული მეთოდის საშუალებით ერთიდან შვიდის ჩათვლით მთელი რიცხვების ფაქტორიალების ითვლის.

დასმული ამოცანის პროგრამული რეალიზაცია ქვემოთ არის წარმოდგენილი, ხოლო მიღებულ შედეგებს შემდეგი სახე აქვს.

შედეგი:



პროგრამული რეალიზაცია:



```
1
2   package rscursion;
3   class Factorial{
4     long fact(int n){
5         if(n==0 || n==1)
6         return 1;
7         else
8             return n*fact(n-1);
9     }
10  }
11
12  public class Rscursion {
13      public static void main(String[] args) {
14        Factorial ob=new Factorial();
15        for(int i=0; i<=7; i++){
16            System.out.println(i + "!=" +ob.fact(i));
17        }
18      }
19  }
```

როდესაც მეთოდი საკუთარი თავის გამოძახებას ახდენს, ახალი ლოკალური ცვლადებისა და პარამეტრებისთვის სტეკში გამოიყოფა ადგილი და მეთოდის კოდი ამ ახალი საწყისი მნიშვნელობებით სრულდება. რეკურსიული მეთოდის ყოველი გამოძახებისას დაბრუნებული ძველი ლოკალური ცვლადები და პარამეტრები სტეკიდან იშლება და შესრულება გრძელდება მეთოდის შიგნით გამოძახების მო-

მენტიდან. ფაქტიურად, რეკურსიული მეთოდები ტელესკოპის მსგავს მოქმედებებს ასრულებენ.

კომპიუტერული რესურსების გადატვირთვის გამო, რაც მეთოდების დამატებით გამოძახებებს უკავშირდება, მრავალი ქვეპროგრამის რეკურსიული ვერსია შესაძლებელია უფრო ნელა შესრულდეს, ვიდრე მათი იტერაციული ანალოგები. მეთოდზე მრავალჯერადმა მიმართვამ შეიძლება სტეკის გადავსება გამოიწვიოს. ვიცით, რომ პარამეტრები და ლოკალური ცვლადები ამ დროს სტეკში ინახება და ამასთან, ყოველი ახალი გამოძახება ამ მონაცემების ახალ ასლებს წარმოშობს. ეს კი დიდ ალბათობას ქმნის იმისას, რომ სტეკი გადაივსოს. რეკურსიული მეთოდების ძირითადი უპირატესობა იმაში მდგომარეობს, რომ მათი გამოყენება იმ ალგორითმების ვერსიების შესაქმნელადაა შესაძლებელი, რომლებიც რეკურსიული გზით უფრო ნათლად გამოისახება, ვიდრე მათი იტერაციული ანალოგები. რეკურსიული მეთოდების გამოყენებისას უნდა ვიზრუნოთ, რომ პროგრამაში სადმე აუცილებლად იქნას გამოყენებული **if** ოპერატორი, რომელიც რეკურსიული მეთოდიდან შედეგის დაბრუნებას რეკურსიული გამოძახების შესრულების გარეშე ახორციელებს. წინააღმდეგ შემთხვევაში, ჩვენ უსასრულო რეკურსიას მივიღებთ.

მაგალითი 2. შევადგინოთ პროგრამა, რომელიც რეკურსიული გზით გამოთვლის ფიბონაჩის რიცხვითი მიმდევრობის პირველი n რაოდენობის წევრების მნიშვნელობებს. აღნიშნული მიმდევრობა იწყება ნულითა და ერთით და ყოველი მომდევნო წევრი წინა ორი წევრის ჯამის ტოლია. ფიბონაჩის რიცხვით მიმდევრობას აქვს ქვემოთ წარმოდგენილი სახე:

0 1 1 2 3 5 8 13 21 34 55...

და ის ბუნებაში სპირალის ფორმას აღწერს.

პროგრამული რეალიზაცია:

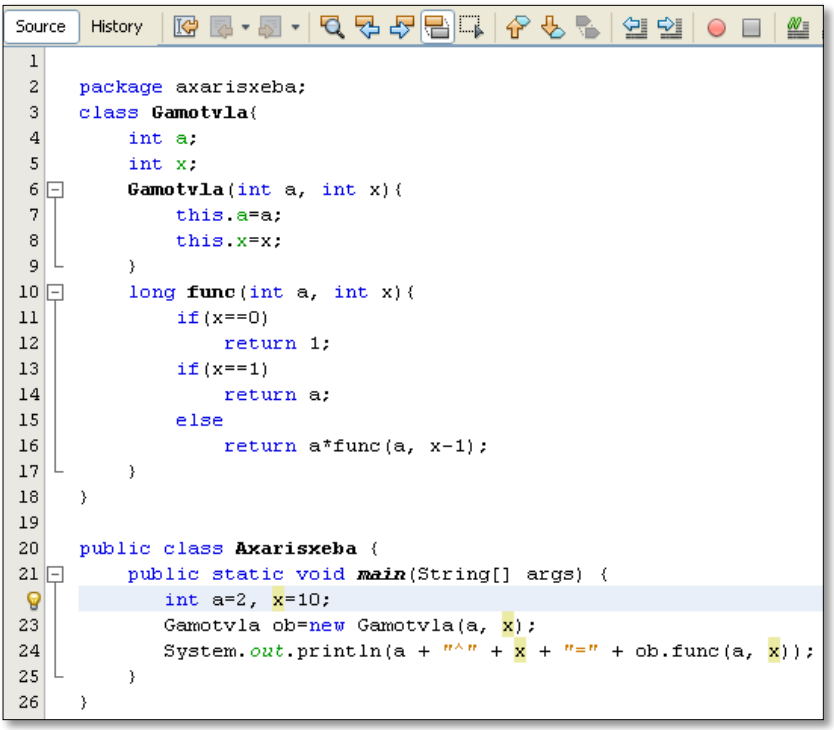
Source History

```
1
2   package fibonacci;
3   class Fibo{
4       long fib(int n){
5           if(n==0 || n==1)
6               return n;
7           else
8               return fib(n-1)+fib(n-2);
9       }
10  }
11  public class Fibonacci {
12      public static void main(String[] args) {
13       Fibo ob=new Fibo();
         for(int i=0; i<11; i++)
15           System.out.print("\t" + ob.fib(i));
16                System.out.println("\nFinish");
17
18
19      }
20  }
21
```

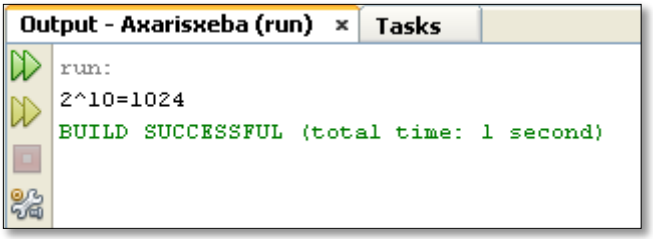
შედეგი:

მაგალითი 3. შევადგინოთ პროგრამა, რომელიც რეკურსიული გზით წარმოგვიდგენს ახარისხების ფუნქციას.

პროგრამული რეალიზაცია:



შედეგი:



<u>დავალება</u>

1. შეადგინეთ პროგრამა, რომელიც რეკურსიული მეთოდის გამოყენებით განსაზღვრავს ორი ნატურალური რიცხვის უდიდესი საერთო გამყოფის მნიშვნელობას.
2. შეადგინეთ პროგრამა, რომელიც რეკურსიული მეთოდის გამოყენებით გამოთვლის a=5-დან b=20-მდე მთელი ტიპის რიცხვითი მონაცემების ჯამს.
3. შეადგინეთ პროგრამა, რომელიც რეკურსიული მეთოდის გამოყენებით გამოთვლის m=1-დან n=10-მდე მთელი ტიპის რიცხვითი მონაცემების ნამრავლს.
4. შეადგინეთ პროგრამა, რომელიც რეკურსიული მეთოდების გამოყენებით გამოთვლის $y = a^x$ გამოსახულების მნიშვნელობას, თუ a და x კენტი რიცხვებია, წინაღმდეგ შემთხვევაში $y = a! - x$ გამოსახულების მნიშვნელობას.
5. შეადგინეთ პროგრამა, რომელიც რეკურსიული მეთოდის გამოყენებით გამოთვლის ფიბონაჩის რიცხვითი მიმდევრობის მე-15 წევრის მნიშვნელობას.

ლაბორატორიული სამუშაო №10

კლასის წევრებზე წვდომის მართვა. კლასის სტატიკური წევრები

ცნობილია, რომ ინკაფსულაცია მონაცემებს კოდთან აკავშირებს. ამას გარდა, ის კიდევ ერთ მნიშვნელოვან ატრიბუტს წარმოგვიდგენს: **წვდომის მართვას**, ანუ ინკაფსულაცია საშუალებას გვაძლევს ვმართოთ, თუ პროგრამის რა ნაწილებს ექნებათ წვდომა კლასის წევრებზე. მაგალითად, თუ კლასის მონაცემებზე წვდომა ექნებათ მხოლოდ მკაცრად განსაზღვრულ მეთოდების ნაკრებს, მაშინ თავიდან ავიცილებთ აღნიშნული მონაცემების „ბოროტად“ გამოყენებას. ამგვარად, როდესაც კლასი სწორადაა რეალიზებული, ის ქმნის ერთგვარ „შავ ყუთს“, რომლის შიდა მექანიზმი დაცულია დაზიანებისგან და ამასთან, ის შეგვიძლია გამოვიყენოთ.

კლასის წევრებზე წვდომას განსაზღვრავს **წვდომის მოდიფიკატორები**, რომლებიც მათ გაცხადებას თან ახლავს. Java ენის წვდომის მოდიფიკატორებია: **public (ღია), private (დახურული)** და **protected (დაცული).** Java ასევე განსაზღვრავს სტანდარტული წვდომის დონესაც (მოდიფიკატორის ცხადი სახით მიუთითებლობის შემთხვევაში).

protected (დაცული) მოდიფიკატორი მხოლოდ მემკვიდრეობითობის შემთხვევაში გამოიყენება.

ახლა კი შევუდგეთ **public** (ღია) და **private** (დახურული) მოდიფიკატორების განსაზღვრას. როდესაც კლასის წევრზე **public** წვდომის მოდიფიკატორია გამოყენებული, ის ნებისმიერი სხვა კოდისთვის წვდომადი ხდება. როდესაც კლასის წევრი მითითებულია, როგორც **private**, ის წვდომადია მხოლოდ იმავე კლასის სხვა წევრებისთვის. ახლა, თქვენთვის გასაგები და ნათელი გახდა, თუ რატომ უსწრებს წინ **main()** მეთოდს **public** მოდიფიკატორი. აღნიშნულ მეთოდს იძახებს კოდი, რომელიც განთავსებულია მოცემული პროგრამის გარეთ.

წვდომის მოდიფიკატორის მიუთითებლობის შემთხვევაში, კლასის წევრი საკუთარი პაკეტის ფარგლებში (შიგნით) ღიად ითვლება, მაგრამ ის მიუწვდომელია კოდისთვის, რომელიც ამ პაკეტის ფარგლებს გარეთაა წარმოდგენილი. როგორც წესი, სასურველია კლასის მონაცემებზე თავისუფალი წვდომა შევზღუდოთ და ეს უკანასკნელი მხოლოდ კლასის მეთოდების საშუალებით განვახორციელოთ. მოდიფიკატორი კლასის წევრის ტიპის ყველა სპეციფიკაციას წინ უსწრებს, ანუ კლასის წევრის გაცხადების ოპერატორი წვდომის მოდიფიკატორით უნდა იწყებოდეს. მაგალითად:

public int x;

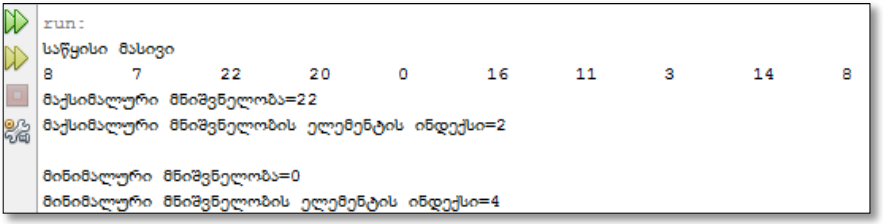
private double y;

private int myMethod( ){ //… }

იმისათვის, რომ გასაგები გახდეს ღია და დახურული წვდომის გამოყენების გავლენა, განვიხილოთ შემდეგი **მაგალითი 1.** შევადგინოთ პროგრამა, რომელიც მთელრიცხვა int a[10] მასივის ელემენტებს მიანიჭებს ნებისმიერ მნიშვნელობებს და განსაზღვრავს მასივის უდიდესი და უმცირესი მნიშვნელობის ელემენტებს და მათ ინდექსებს. კლასის წევრებზე გამოვიყენოთ წვდომის მოდიფიკატორები.

ქვემოთ წარმოდგენილია დასმული ამოცანის გადაწვეტის შედეგი და შემდგომ მისი პროგრამული რეალიზაცია.

შედეგი:



პროგრამული რეალიზაცია:

```
1  package access1;
2  import java.util.Random;
3  class student{
4      public int a[]=new int[10];
5      private int maxi, mini, k, p;
6      student(){
7          Random rand=new Random();
8          for(int i=0; i<a.length; i++){
9              a[i]=(int)(25*rand.nextDouble());
10             System.out.print(a[i] + "\t");}}
11     public void method1(){
12         maxi=a[0]; k=0;
13          for(int i=0; i<a.length; i++){
14              if(maxi<=a[i]){
15                  maxi=a[i];
16                  k=i;}}
17          System.out.println("\nმაქსიმალური მნიშვნელობა=" + maxi);
18          System.out.println("მაქსიმალური მნიშვნელობის ელემენტის ინდექსი=" + k);}
19
20     public void method2(){
21          mini=a[0]; p=0;
22          for(int i=0; i<a.length; i++){
23              if(mini>=a[i]){
24                  mini=a[i];
25                  p=i;}}
26          System.out.println("\nმინიმალური მნიშვნელობა=" + mini);
27          System.out.println("მინიმალური მნიშვნელობის ელემენტის ინდექსი=" + p);}}
28 public class Access1 {
29     public static void main(String[] args) {
30         System.out.println("საწყისი მასივი");
31         student object1=new student();
32         object1.method1();
33         object1.method2();
34         //object1.k; დაუშვებელი ოპერაციაა!
35     }}
```

როგორც წესი, მეთოდები უზრუნველყოფენ ხოლმე თავისივე კლასის მონაცემებზე წვდომას, თუმცა ეს სავალდებულო არ არის.

ზოგ შემთხვევაში, სასურველია განვსაზღვროთ კლასის წევრი, რომელიც მოცემული კლასის ნებისმიერი ობიექტის-

გან დამოუკიდებლად გამოიყენება. კლასის ასეთი წევრის შესაქმნელად აუცილებელია მის გამოცხადებას წინ დავურთოთ **static** საკვანძო სიტყვა. როდესაც კლასის წევრი ცხადდება როგორც სტატიკური (**static**), ის მისაწვდომი ხდება თავისი კლასის ნებისმიერი ობიექტის შექმნამდე. სტატიკური შესაძლებელია იყოს როგორც მეთოდები, ასევე ცვლადები. სტატიკური წევრის ყველაზე თვალსაჩინო მაგალითს **main()** მეთოდი წარმოადგენს. აღნიშნული მეთოდი სტატიკურია, რადგან ის ნებისმიერი ობიექტის შექმნამდე უნდა იქნას გაცხადებული. ეგზემპლარის სტატიკური ცვლადები გლობალურ ცვლადებს წარმოადგენს და მათი კლასის ობიექტების გამოცხადებისას პროგრამა არ ქმნის სტატიკური ცვლადების ასლებს. შესაბამისად, კლასის ყველა ეგზემპლარი ერთსა და იმავე სტატიკურ ცვლადს ერთობლივად გამოიყენებს.

სტატიკური მეთოდების გამოყენების დროს შემდეგი შეზღუდვები უნდა გავითვალისწინოთ:

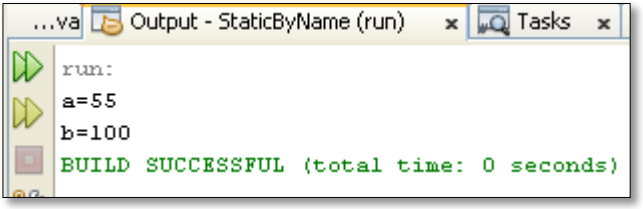
- მათ უშუალოდ, მხოლოდ სხვა სტატიკური მეთოდების გამოძახება შეუძლიათ;
- უშუალო წვდომა მათ მხოლოდ სტატიკურ ცვლადებზე გააჩნიათ;
- მათ არ შეუძლიათ this და super ტიპის წევრებზე მიმართვა.

სტატიკური მეთოდები და ცვლადები, თავიანთი კლასის გარეთ შეგვიძლია ობიექტებისგან დამოუკიდებლად გამოვიყენოთ. ამისათვის საკმარისია კლასის სახელის მითითება, რომელსაც წერტილის ოპერატორი უნდა მოყვებოდეს. მაგალითად, თუ სტატიკური მეთოდი კლასის გარეთ გვსურს გამოვიძახოთ, საჭიროა შემდეგი ზოგადი ფორმის გამოყენება:

კლასის_სახელი.მეთოდი();

აქ კლასის_სახელის ქვეშ იმ კლასის სახელი იგულისხმება, რომელშიც სტატიკური მეთოდია გამოცხადებული. ანალოგიურად ვიქცევით სტატიკური ცვლადების შემთხვევაშიც. სწორედ, ამ გზით ხდება Java-ში გლობალური მეთოდებისა და ცვლადების მმართველი ვერსიების რეალიზება. ზემოაღნიშნულის საილუსტრაციოდ განვიხილოთ შემდეგი **მაგალითი 2,** სადაც main() მეთოდის შიგნით StaticDemo კლასის სახელისა და წერტილის ოპერატორის გამოყენებით ხორციელდება მიმართვა callme() სტატიკურ მეთოდსა და b სტატიკურ ცვლადზე. ქვემოთ არის წარმოდგენილი ამოცანის გადაწყვეტის შედეგი და შემდგომ მისი პროგრამული რეალიზაცია.

შედეგი:



პროგრამული რეალიზაცია:

```
Source  History
1
2   package staticbyname;
3   class StaticDemo{
4       static int a=55;
5       static int b=100;
6       static void callme(){
7           System.out.println("a=" +a);
8       }
9   }
10  public class StaticByName {
11      public static void main(String[] args) {
12          StaticDemo.callme();
13          System.out.println("b=" + StaticDemo.b);
14      }
15  }
```

მაგალითი 3. შევადგინოთ პროგრამა, რომელიც კლასის სტატიკური მეთოდის საშუალებით წარმოგვიდგენს ერთიდან 10-ის ჩათვლით რიცხვითი მონაცემების გამრავლების ტაბულას.

შედეგი:

```
run:
        1   2   3   4   5   6   7   8   9   10
1       1   2   3   4   5   6   7   8   9   10
2       2   4   6   8   10  12  14  16  18  20
3       3   6   9   12  15  18  21  24  27  30
4       4   8   12  16  20  24  28  32  36  40
5       5   10  15  20  25  30  35  40  45  50
6       6   12  18  24  30  36  42  48  54  60
7       7   14  21  28  35  42  49  56  63  70
8       8   16  24  32  40  48  56  64  72  80
9       9   18  27  36  45  54  63  72  81  90
10      10  20  30  40  50  60  70  80  90  100
გამრავლების ტაბულა:
```

პროგრამული რეალიზაცია:

```
1  package table1;
2  class student{
3      student(){
4          System.out.println("გამრავლების ცხრილი:");
5      }
6      static void method1(){
7          for (int i=1;i<=10;i++) System.out.print("\t"+i);
8          System.out.println();
9          for (int i=1;i<=10;i++)
10         {
11          System.out.print(i+"\t");
12           for (int j=1;j<=10;j++)
13             {
14              System.out.print(i*j+"\t");
15                          }
16              System.out.println();
17                 } }}
18 public class Table1 {
19     public static void main(String[] args) {
20       student.method1(); //სტატიკური მეთოდის გამოძახება ობიექტის შექმნამდე
21       student object1=new student();
22     }
23 }
```

როგორც ვხედავთ, პროგრამაში ჩვენ არ შეგვიქმნია მოცემული კლასის არცერთი ობიექტი (ცხადია, აქ ვიგულისხმეთ, კლასის სტატიკური მეთოდის გამოძახებამდე) და სტატიკურ მეთოდზე მიმართვა საკუთარი კლასის გარეთ განვახორციელეთ კლასის სახელისა და წერტილის ოპერატორის საშუალებით.

დავალება

1. შეადგინეთ პროგრამა, რომელიც კლასის სტატიკური მეთოდის გამოყენებით 0-დან 10000-მდე იპოვის ყველა ისეთ რიცხვს, რომლის ციფრების ჯამი ამავე ციფრების ნამრავლის ტოლი იქნება.

2. შეადგინეთ პროგრამა, რომელიც კლასის სტატიკური მეთოდის გამოყენებით ერთიდან 10000-მდე იპოვის ყველა წესიერ რიცხვს, რაც გულისხმობს იმ ფაქტს, რომ რიცხვის გამყოფების ჯამი ამავე რიცხვის ტოლი უნდა იყოს.

3. შეადგინეთ პროგრამა, რომელიც კლასის წევრებზე წვდომის მოდიფიკატორების გამოყენებით 20-ელემენტიან მთელრიცხვა მასივში გამოთვლის კენტინდექსიანი და ამავდროულად, კენტი მნიშვნელობის მქონე ელემენტების საშუალო არითმეტიკულს.

4. შეადგინეთ პროგრამა, რომელიც კლასის წევრებზე წვდომის მოდიფიკატორების გამოყენებით 4x4 განზომილების მქონე კვადრატულ მატრიცაში ცალ-ცალკე დათვლის კენტი და ლუწი მნიშვნელობის ელემენტების რაოდენობას და მიღებულ შედეგებს ერთმანეთს შეადარებს.

ლაბორატორიული სამუშაო №11

String კლასი. ცვლადი რაოდენობის არგუმენტებიანი მეთოდები და მათი გადატვირთვა

String კლასი Java–ს კლასების ბიბლიოთეკის ყველაზე ხშირად გამოყენებადი კლასია. ამის მიზეზი ის გახლავთ, რომ სტრიქონები დაპროგრამების უმნიშვნელოვანესი ელემენტია.

პირველ რიგში, საჭიროა გავაცნობიეროთ, რომ ნებისმიერი სტრიქონი String კლასის ობიექტს წარმოადგენს. სტრიქონული კონსტანტებიც კი ამავე კლასის ობიექტებია. მაგალითად, ოპერატორში:

System.out.println("ესეც String კლასის ობიექტია");

სტრიქონი – **„ესეც String კლასის ობიექტია"** – რეალურად წარმოადგენს String კლასის ობიექტს.

მეორეს მხრივ, String კლასის ობიექტები უცვლელია, რაც იმას გულისხმობს, რომ აღნიშნული კლასის ობიექტის შექმნის შემდეგ, მისი შიგთავსის ცვლილება შეუძლებელია. ყოველივე ზემოთ თქმული შეიძლება სერიოზულ შეზღუდვად მოგვეჩვენოს, მაგრამ ორი მიზეზიდან გამომდინარე, რეალურად სხვა სურათი იქმნება:

- თუ სტრიქონის შეცვლაა საჭირო, ყოველთვის შეიძლება ახალი სტრიქონის შექმნა, რომელიც ცვლილებებს მოიცავს;
- Java-ში განსაზღვრულია StringBuffer კლასი, რომელიც String კლასის თანასწორია და ის საშუალებას იძლევა შევცვალოთ სტრიქონები, რაც თავის მხრივ, უფლებას გვაძლევს ყველა ჩვეულებრივი მანიპულაცია განვახორციელოთ სტრიქონებზე.

სტრიქონების შექმნის მრავალი გზა არსებობს. მათ შორის ყველაზე მარტივია შემდეგი სახის ოპერატორის გამოყენება:

String myString="ტექსტური სტრიქონი";

String კლასის ობიექტის შექმნისთანავე მისი გამოყენება შესაძლებელია ყველა სიტუაციაში, რომელშიც სტრიქონებთან მუშაობაა დასაშვები. მაგალითად, შემდეგი ოპერატორი წარმოგვიდგენს myString ობიექტის შიგთავსს:

System.out.println(myString);

String კლასის ობიექტებისთვის Java-ში განსაზღვრულია „+" ოპერატორი, რომელიც ორი სტრიქონის გაერთიანებას ასრულებს. მაგალითად, შემდეგი ოპერატორი:

String myString="მე" + „მომწონს" + „Java.";

myString ცვლადს ანიჭებს შემდეგი სტრიქონის მნიშვნელობას: „მე მომწონს Java”.

ახლა კი განვიხილოთ, ზემოთ წარმოდგენილი კონცეფციების ამსახველი **მაგალითი 1**, რომელიც ორი სტრიქონის გაერთიანებას ახდენს და კონსოლზე საწყისი და მიღებული სტრიქონები გამოაქვს.

პროგრამული რეალიზაცია:

```
1   package string1;
2   public class String1 {
3       public static void main(String[] args) {
4           String ob1="მე ქართველი ვარ ";
5           String ob2="და მაშასადამე, ვარ ევროპელი.";
6           String ob3=ob1+ob2;
7           System.out.println(ob1);
8           System.out.println(ob2);
9           System.out.println(ob3);
10      }
11  }
```

შედეგი:

```
Output - String1 (run)
run:
მე ქართველი ვარ
და მაშასადამე, ვარ ევროპელი.
მე ქართველი ვარ და მაშასადამე, ვარ ევროპელი.
```

String კლასი რამდენიმე მეთოდს მოიცავს. გავეცნოთ ზოგიერთ მათგანს. **equals()** მეთოდით შესაძლებელია ორი სტრიქონის ტოლობაზე შემოწმება. **length()** მეთოდი სტრიქონის სიგრძეს განსაზღვრავს. **charAt()** მეთოდით მითითებული ინდექსით სიმბოლოს მიღებაა შესაძლებელი. ქვემოთ წარმოდგენილია ამ მეთოდების ჩაწერის ზოგადი ფორმები:

boolean aquals(მეორე სტრიქონი);

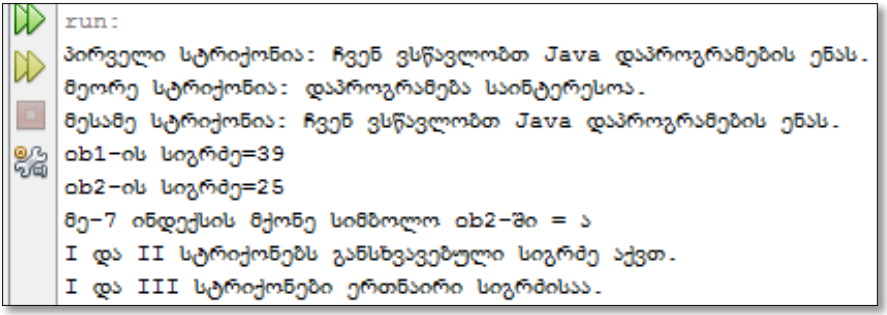
int length();

char charAt(ინდექსი);

მაგალითი 2. შევადგინოთ პროგრამა, რომელიც ახდენს სტრიქონების ტოლობაზე შემოწმებას, მათი სიგრძის განსაზღვრას და მითითებული ინდექსით სტრიქონიდან შესაბამისი სიმბოლოს ამოღებას.

ქვემოთ წარმოდგენილია დასმული ამოცანის გადაწყვეტის შედეგები, ხოლო შემდგომ მისი პროგრამული რეალიზაცია.

შედეგი:



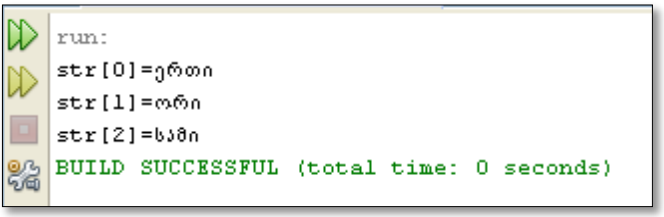
პროგრამული რეალიზაცია:

```
package string1;
public class String1 {
    public static void main(String[] args) {
        String ob1="ჩვენ ვსწავლობთ Java დაპროგრამების ენას.";
        String ob2="დაპროგრამება საინტერესოა.";
        String ob3=ob1;
        System.out.println("პირველი სტრიქონია: " +ob1);
        System.out.println("მეორე სტრიქონია: " +ob2);
        System.out.println("მესამე სტრიქონია: " +ob3);
        System.out.println("ob1-ის სიგრძე=" + ob1.length());
        System.out.println("ob2-ის სიგრძე=" + ob2.length());
     System.out.println("მე-7 ინდექსის მქონე სიმბოლო ob2-ში = " +ob2.charAt(7));
        if(ob1.equals(ob2))
        System.out.println("I და II სტრიქონები ერთნაირი სიგრძისაა.");
        else
            System.out.println("I და II სტრიქონებს განსხვავებული სიგრძე აქვთ.");
        if(ob1.equals(ob3))
        System.out.println("I და III სტრიქონები ერთნაირი სიგრძისაა.");
        else
            System.out.println(" I და III სტრიქონებს განსხვავებული სიგრძე აქვთ.");
    }
}
```

ახლა წარმოვაგინოთ სტრიქონთა მასივის ამსახველი მარტივი პროგრამა:

```
package stringdemo;
public class StringDemo {
    public static void main(String[] args) {
    String str[]={ "ერთი", "ორი", "სამი"};
    for(int i=0; i<str.length; i++)
        System.out.println("str[" + i + "]=" + str[i]);
    }
}
```

შედეგი:



JDK5-ში დამატებულ იქნა ახალი ფუნქციონალური შესაძლებლობა, რომელმაც მნიშვნელოვნად გაამარტივა ცვლადი რაოდენობის არგუმენტების შემცველი მეთოდების შექმნა. ამ საშუალებას **vararg** (შემოკლებით, variable-length arguments – ცვლადი სიგრძის არგუმენტების სია) ეწოდა. მეთოდს, რომელიც არგუმენტების ცვლად რაოდენობას იღებს, **ცვლადი რაოდენობის არგუმენტებიან მეთოდს** უწოდებენ. მსგავსი მეთოდები პროგრამებში ხშირად გვხვდება. ცვლადი სიგრძის არგუმენტების სიის მისათითებლად სამი წერტილი (. . .) გამოიყენება. მაგალითად, vaTest() მეთოდი ცვლადი სიგრძის არგუმენტების სიის გამოყენებით შეიძლება შემდეგი ფორმით ჩავწეროთ:

static void vaTest(int … v) {

ეს სინტაქსური კონსტრუქცია კომპილატორს მიუთითებს, რომ vaTest() მეთოდი შეიძლება გამოძახებულ იქნას უარგუმენტებოდ ან რამდენიმე არგუმენტით. შედეგად, v[] მასივი არაცხადი სახით აღიწერება როგორც int []. ამგვარად,

vaTest() მეთოდში v[] მასივზე წვდომა ჩვეულებრივი მასივის სინტაქსის გამოყენებით ხდება.

ცვლადი რაოდენობის არგუმენტების შესაბამის პარამეტრთან ერთად მეთოდი „ნორმალურ“ პარამეტრებსაც შეიძლება შეიცავდეს. ამ შემთხვევაში, ცვლადი რაოდენობის არგუმენტების შესაბამისი პარამეტრი მეთოდის გამოცხადებისას ბოლო პარამეტრს უნდა წარმოადგენდეს. მაგალითად, სავსებით დასაშვებია მეთოდის შემდეგი ფორმით გამოცხადება:

int vaTest (int a, int b, double c, int … vals) {

ამ შემთხვევაში, **vaTest()** მეთოდის პირველი სამი არგუმენტი პირველ სამ პარამეტრს შეესაბამება, ხოლო ყველა დანარჩენი არგუმენტი – vals პარამეტრს. საჭიროა გვახსოვდეს, რომ ცვლადი რაოდენობის არგუმენტების შესაბამისი პარამეტრი მეთოდში ბოლოს უნდა იყოს მითითებული.

ცვლადი რაოდენობის არგუმენტების შესაბამისი მეორე პარამეტრის მითითება მეთოდში დაუშვებელია!

მაგალითი 3. შევადგინოთ პროგრამა, რომელიც ცვლადი არგუმენტების შემცველი მეთოდის გამოყენებით გამოთვლის საკუთარი არგუმენტების ჯამსა და ნამრავლს.

პროგრამული რეალიზაცია:

```
1   package vars;
2   public class main {
3   static int s, p;
4        static void method1(int ... v){
5            s=0; p=1;
6            for(int x : v){
7                System.out.print(x+ "\t");
8                s+=x;
9                p*=x;
10           }
11           System.out.println("\ns=" + s);
12           System.out.println("p=" + p);
13           System.out.println();
14       }
15      public static void main(String[] args) {
16         method1(1, 2, 3, 4, 5);
17         method1(1, 2, 3, 4, 5, 6);
18      }
19  }
```

შედეგი:

```
Output - Vars (run)
run:
1       2       3       4       5
s=15
p=120

1       2       3       4       5       6
s=21
p=720
```

მეთოდი, რომელიც იღებს ცვლადი სიგრძის არგუმენტების სიას, შეიძლება გადატვირთულ იქნას. უნდა გვახსოვდეს რომ (...) კონსტრუქცია კომპილატორს აიძულებს პარამეტრი დაამუშავოს როგორც მოცემული ტიპის მასივი. ასეთ სიტუაციებში გადატვირთული მეთოდის საჭირო ვარიანტის განსაზღვრის მიზნით Java ტიპებს შორის განსხვავებას იყენებს.

მაგალითი 4. შევადგინოთ ცვლადი არგუმენტების შემცველი მეთოდის გადატვირთვის ამსახველი პროგრამა, სადაც მეთოდის საწყისი ვერსია ითვლის საკუთარი მთელრიცხვა არგუმენტების ნამრავლს, ხოლო მეორე ვერსია - ნამდვილრიცხვა არგუმენტების კვადრატების ჯამს.

ქვემოთ წარმოდგენილია დასმული ამოცანის გადაწყვეტის შედეგები, ხოლო შემდგომ მისი პორგამული რეალიზაცია.

შედეგი:

```
run:
1       2       3       4       5       6       7
ნამრავლი=5040
1.5     2.3     3.6     4.1
კვადრატების ჯამი=37.31
```

პროგრამული რეალიზაცია:

```
1   package passarray;
2    class student{
3        static int p;
4        static double s;
5        static void method1(int ... v){
6            p=1;
7           for(int x: v){
8               System.out.print(x + "\t");
9               p*=x;
10          }
11           System.out.println("\nნამრავლი=" + p);
12      }
13      static void method1(double ... v){
14           s=0;
15          for(double x: v){
16              System.out.print(x + "\t");
17              s+=Math.pow(x, 2);
18          }
19           System.out.println("\nკვადრატების ჯამი=" + s);
20      }
21   }
22   class PassArray {
23      public static void main(String[] args) {
24           student ob1=new student();
          ob1.method1(1, 2, 3, 4, 5, 6, 7);
          ob1.method1(1.5, 2.3, 3.6, 4.1);
27      }
28      }
```

დავალება

1. შეადგინეთ პროგრამა, რომელიც მოახდენს სამი სტრიქონის გაერთიანებას, განსაზღვრავს თვითოეული სტრიქონის სიგრძეს და ბოლო ორ სტრიქონს ერთმანეთს შეადარებს.
2. შეადგინეთ ცვლადი არგუმენტების შემცველი მეთოდის გადატვირთვის ამსახველი პროგრამა, სადაც მეთოდის საწყისი ვერსია გამოითვლის საკუთარი მთელრიცხვა შვიდი არგუმენტის საშუალო არითმეტიკულს, ხოლო მეორე ვერსია - ექვსი ნამდვილრიცხვა არგუმენტის ჯამიდან კვადრატული ფესვის მნიშვნელობას.
3. შეადგინეთ პროგრამა, რომელიც ცვლადი არგუმენტების შემცველი მეთოდის გამოყენებით გამოთვლის მხოლოდ ლუწი მნიშვნელობის არგუმენტების ნახევარ-ჯამს. მეთოდი გამოიძახეთ ჯერ ხუთი და შემდგ რვა არგუმენტით.
4. შეადგინეთ პროგრამა, რომელიც ცვლადი არგუმენტების შემცველი მეთოდის გამოყენებით გამოთვლის მხოლოდ კენტინდექსიანი მნიშვნელობის არგუმენტების კუბების ჯამს. მეთოდი გამოიძახეთ ჯერ ოთხი და შემდგ ათი არგუმენტით.

ლაბორატორიული სამუშაო №12

მემკვიდრეობითობა. კლასის წევრებზე წვდომა მემკვიდრეობითობის დროს

ობიექტზე ორიენტირებული დაპროგრამების ერთ–ერთ ფუნდამენტურ ცნებას **მემკვიდრეობითობა** წარმოადგენს. ის საშუალებას გვაძლევს შევქმნათ იერარქიული კლასიფიკაციები. მემკვიდრეობითობის გამოყენებით შეიძლება შეიქმნას საერთო კლასი, რომელიც განსაზღვრავს ერთმანეთთან დაკავშირებული ელემენტების ნაკრების საერთო მახასიათებლებს. შემდგომ ეს კლასი მემკვიდრეობით შეიძლება მიიღოს გაცილებით უფრო სპეციალიზირებულმა კლასებმა, რომელთაგან თვითოეულ მათგანს თავისი უნიკალური მახასიათებლების დამატება შეეძლება. Java-ს ტერმინოლოგიაში მშობელ კლასს **სუპერ კლასი** ეწოდება, ხოლო მემკვიდრე კლასს – **ქვეკლასი**. შესაბამისად, ქვეკლასი სუპერ–კლასის სპეციალიზირებული ვერსიაა. ის მემკვიდრეობით იღებს სუპერ–კლასის მიერ განსაზღვრულ ყველა წევრს და უმატებს საკუთარ, უნიკალურ ელემენტებს.

მემკვიდრეობითობის განსახორციელებლად საკმარისია ერთი კლასის განსაზღვრა მეორე კლასში მოვათავსოთ **extends** საკვანძო სიტყვის გამოყენებით.

ის ფაქტი, რომ ესა თუ ის კლასი სუპერ კლასს წარმოადგენს თავისი ქვეკლასისთვის, არ ნიშნავს, რომ მისი დამოუკიდებლად გამოყენება შეუძლებელია. უფრო მეტიც, ქვეკლასი შეიძლება თავის მხრივ, სუპერ კლასსაც წარმოადგენდეს სხვა ქვეკლასისთვის.

ქვეკლასის გამოცხადების ზოგადი ფორმა შემდეგია:

class ქვეკლასის_სახელი extends სუპერ კლასის_სახელი {

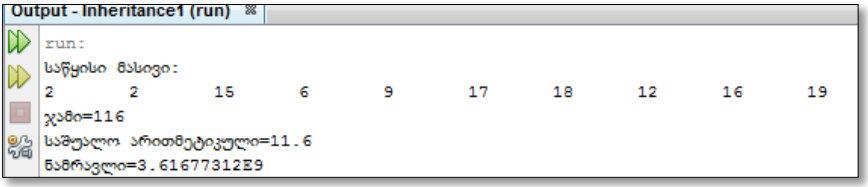
//კლასის ტანი; }

ყოველ ქვეკლასს მხოლოდ ერთი სუპერ კლასი უნდა მიეთითოს. Java-ში დაუშვებელია რამდენიმე სუპერ კლასისგან ერთი ქვეკლასის მემკვიდრეობა.

მაგალითი 1. შევადგინოთ პროგრამა, რომელიც კლასების მემკვიდრეობითობის გამოყენების გზით int a[10] ელემენტიან მასივში გამოთვლის ელემენტების ჯამს, ნამრავლსა და საშუალო არითმეტიკულ მნიშვნელობას.

ქვემოთ წარმოდგენილია დასმული ამოცანის გადაწყვეტის შედეგები და შემდგომ მისი პროგრამული რეალიზაცია.

შედეგი:



პროგრამული რეალიზაცია:

```
package inheritance1;
import java.util.Random;
class student{
    private int s;
    public double p;
    public int a[]=new int[10];
    student(){
      s=0; p=1;
      Random rand=new Random();
      for(int i=0; i<a.length; i++){
          a[i]=(int)(20*rand.nextDouble());
          System.out.print(a[i]+ "\t");}}
    public void method1(){
        for(int x : a)
            s+=x;
        System.out.println("\nჯამი=" +s);
        System.out.println("საშუალო არითმეტიკული=" + (s/10.));
    }}
    class student1 extends student{
        public void method2(){
            for(int x : a)
                p*=x;
            System.out.println("ნამრავლი=" + p);}}

   public class Inheritance1 {
    public static void main(String[] args) {
        System.out.println("საწყისი მასივი:");
      student1 ob1=new student1();
      ob1.method1();
      ob1.method2();}}
```

მართლია, ქვეკლასი სუპერ კლასის ყველა წევრს თავის თავში მოიცავს, მაგრამ სუპერ კლასის დახურულ (**private**) წევრებზე მას არ აქვს წვდომა.

განვიხილოთ კლასების მარტივი იერარქია.

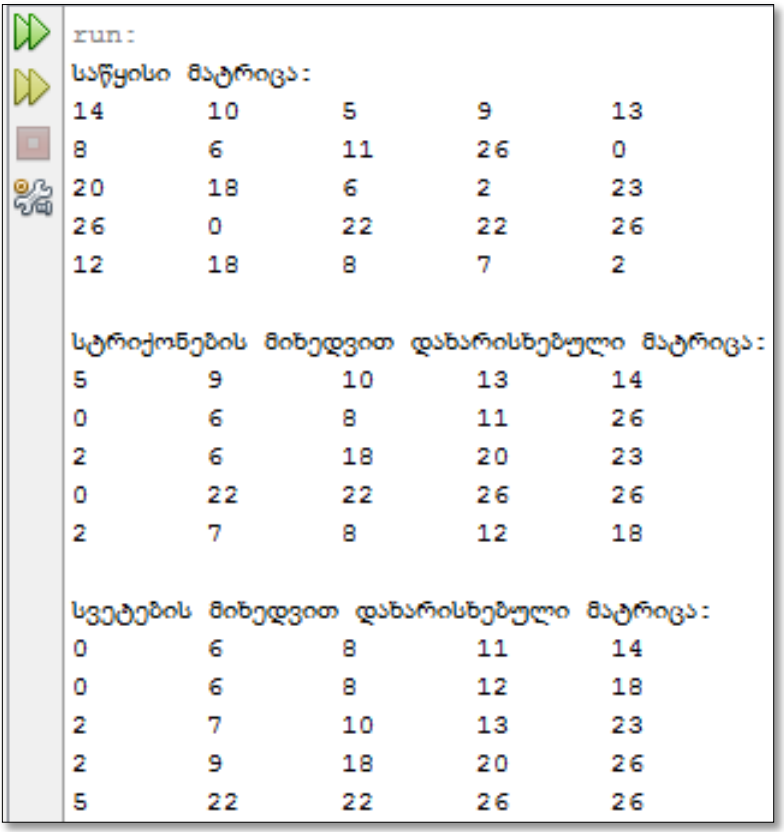
```
1
2   package simpleinheritance;
3   class A{
4       int i;
5       private int j;
6       void setij(int x, int y){
7      i=x;
8      j=y;
9   }
10  }
11   class B extends A{
12       int total;
13       void sum(){
14          total=i+j; // შეცდომა: j ამ კლასშია არ არის წვდომადი.
15       }
16   }
17  public class SimpleInheritance {
18      public static void main(String[] args) {
19
20          B subOb=new B();
21          subOb.setij(10, 12);
22          subOb.sum();
23          System.out.println(" ჯამი=" + subOb.total);
24
25      }
26  }
```

წარმოდგენილი პროგრამის კომპილაცია შეუძლებელია, რადგან j ცვლადის გამოყენება B კლასის sum() მეთოდში იწვევს კლასის წევრებზე წვდომის წესების დარღვევას. რადგან j ცვლადი გაცხადებულია როგორც, private (დახურული), ამიტომ ის წვდომადია მხოლოდ თავისი კლასის სხვა წევრებისთვის. ქვეკლასს მასზე წვდომა არ გააჩნია. კლასის დახურული წევრი დახურულ წევრად რჩება თავისი კლასისთვის

და საკუთარი კლასის გარეთ ის არცერთი კოდისთვის (მათ შორის, ქვეკლასებისთვის) არ არის წვდომადი.

მაგალითი 2. შევადგინოთ პროგრამა, რომელიც კლასების მემკვიდრეობითობის გამოყენების გზით int a[5][5] კვადრატულ მატრიცას ჯერ სტრიქონების, ხოლო შემდეგ სვეტების მიხედვით დაახარისხებს ზრდადობით.

შედეგი:



პროგრამული რეალიზაცია:

```
package matrixsort;
import java.util.Random;
class student{
    public int a[][]=new int[5][5];
    student(){
     Random rand=new Random();
     for(int i=0; i<5; i++)
         for(int j=0; j<5; j++)
             a[i][j]=(int)(30*rand.nextDouble());}
    public void method2(){
        for(int k=1; k<25; k++){
        for(int i=0; i<5; i++){
            for(int j=0; j<4; j++){
           if(a[i][j]>a[i][j+1]){int temp=a[i][j];
               a[i][j]=a[i][j+1];
               a[i][j+1]=temp;}}}}}
  class student1 extends student{
      public void method1(){
      for(int i=0; i<5; i++){
         for(int j=0; j<5; j++)
              System.out.print(a[i][j]+ "\t");
          System.out.println();}}
      public void method3(){
          for(int k=1; k<25; k++){
       for(int j=0; j<5; j++){
            for(int i=0; i<4; i++){
           if(a[i][j]>a[i+1][j]){
               int temp=a[i][j];
               a[i][j]=a[i+1][j];
               a[i+1][j]=temp; }}}}}
public class MatrixSort {
    public static void main(String[] args) {
        System.out.println("საწყისი მატრიცა:");
       student1 ob1=new student1();
       ob1.method1();
       System.out.println("\nსტრიქონების მიხედვით დახარისხებული მატრიცა:");
       ob1.method2();
       ob1.method1();
       System.out.println("\nსვეტების მიხედვით დახარისხებული მატრიცა:");
       ob1.method3();
       ob1.method1();}}
```

დავალება

1. შეადგინეთ პროგრამა, რომელიც კლასების მემკვიდრეობითობის გამოყენების გზით int b[15] ელემენტიან მასივში გამოთვლის ლუწი მნიშვნელობის ელემენტების ჯამიდან მე-4 ხარისხის ფესვის მნიშვნელობას.

2. შეადგინეთ პროგრამა, რომელიც კლასების მემკვიდრეობითობის გამოყენების გზით int c[17] ელემენტიან მასივში გამოთვლის კენტინდექსიანი ელემენტების საშუალო არითმეტიკულ მნიშვნელობას.

3. შეადგინეთ პროგრამა, რომელიც კლასების მემკვიდრეობითობის გამოყენების გზით int a[7][7] კვადრატულ მატრიცაში სვეტების მიხედვით განსაზღვრავს უმცირესი მნიშვნელობის ელემენტებს და მიღებულ შედეგებს შორის უდიდესს.

4. შეადგინეთ პროგრამა, რომელიც კლასების მემკვიდრეობითობის გამოყენების გზით int a[4][4] კვადრატულ მატრიცაში სტრიქონების მიხედვით ცალ-ცალკე გამოთვლის ელემენტების კვადრატების ჯამთა მნიშვნელობებს და დაბეჭდავს მიღებულ შედეგებს შორის უდიდესი შედეგის მომცემ სტრიქონის ელემენტებს.

ლაბორატორიული სამუშაო №13

super საკვანძო სიტყვის გამოყენება. მრავალდონიანი იერარქიის შექმნა

რიგ შემთხვევებში საჭირო ხდება ისეთი სუპერ კლასის შექმნა, რომლის რეალიზაციის დეტალები წვდომადია მხოლოდ მისთვის (ანუ ადგილი აქვს მონაცემთა დახურული წევრების გამოყენებას). ასეთ დროს ქვეკლასი დამოუკიდებლად (უშუალოდ) ვერ მიმართავს დახურულ ცვლადებს და ვერ ახდენს მათ ინიციალებას. რადგან ინკაფსულაცია ობიექტზე ორიენტირებული დაპროგრამების ერთ–ერთი მთავარი ატრიბუტია, გასაკვირი არ არის, რომ Java ამ პრობლემის გადაწყვეტას გვთავაზობს. ყველა შემთხვევაში, როდესაც ქვეკლასმა უშუალოდ უნდა მიმართოს თავის სუპერ კლასს, შეგვიძლია **super** საკვანძო სიტყვა გამოვიყენოთ. მას ორი ზოგადი ფორმა გააჩნია. პირველი გამოიყენება სუპერ კლასის კონსტრუქტორის გამოსაძახებლად, ხოლო მეორე – სუპერ კლასის იმ წევრზე მიმართვისათვის, რომელიც ქვეკლასის წევრის მიერაა დაფარული. განვიხილოთ ორივე ფორმა.

ქვეკლასს შეუძლია თავისი სუპერ კლასის მიერ განსაზღვრული კონსტრუქტორის გამოძახება **super** საკვანძო სიტყვის შემდეგი ფორმის გამოყენებით:

super(არგუმენტების_სია);

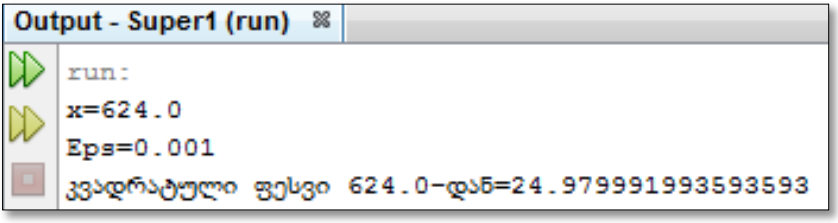
აქ **არგუმენტების_სია** განსაზღვრავს ნებისმიერ არგუმენტს, რომელიც სუპერ კლასის კონსტრუქტორს ესაჭიროება. **super()** მეთოდის გამოძახება ყოველთვის პირველ ოპერატორს უნდა წარმოადგენდეს, რომელიც ქვეკლასის კონსტრუქტორში სრულდება.

super() მეთოდის გამოყენების საილუსტრაციოდ განვიხილოთ **მაგალითი 1,** რომელიც ნამდვილი რიცხვიდან კვადრატული ფესვის მნიშვნელობას Eps სიზუსტით ითვლის შემდეგი იტერაციული ფორმულის საფუძველზე:

$$y_n = \frac{1}{2}\left(y_{n-1} + \frac{x}{y_{n-1}}\right).$$

ქვემოთ წარმოდგენილია დასმული ამოცანის ამოხსნის შედეგი, ხოლო შემდეგომ მისი პროგრამული რეალიზაცია.

შედეგი:



პროგრამული რეალიზაცია:

```
1   package super1;
2   import java.lang.Math;
3   class student{
4       double x, Eps;
5       student(double x, double Eps){
6           this.x=x;
7           this.Eps=Eps; }}
8    class student1 extends student{
9        double y;
10       student1(double x, double Eps, double y){
11      super(x,Eps);
12      this.y=y; }
13     void metho1(){
14         double yp;
15         do{
16             yp=y;
17             y=(yp+x/yp)/2;
18         }while((Math.abs(y-yp))>=Eps);
19         System.out.println("კვადრატული ფესვი " + x + "-დან=" + y);}}
20   public class Super1 {
21      public static void main(String[] args) {
22        double x=624, Eps=0.001, y=1;
23          student1 ob1=new student1(x, Eps, y);
24          System.out.println("x=" + x);
25          System.out.println("Eps=" + Eps);
26          ob1.metho1();}}
```

განვიხილოთ super() მეთოდის გამოყენების ძირითადი კონცეფციები. როდესაც ქვეკლასი super() მეთოდს იძახებს, ამ დროს ის უშუალოდ თავისი სუპერ კლასის კონსტრუქტორის გამოძახებას ახდენს. ამგვარად, super() მეთოდი ყოველთვის მიმართავს სუპერ კლასს, რომელიც კლასების იერარქიულ სტრუქტურაში უშუალოდ გამომძახებელი კლასის

ზემოთ მდებარეობს. ეს სამართლიანია მრავალდონიანი იერარქიის შემთხვევაშიც. ამასთან, super() მეთოდი ყოველთვის პირველი შესასრულებელი ოპერატორი უნდა იყოს ქვეკლასის კონსტრუქტორში.

super საკვანძო სიტყვის გამოყენების მეორე ფორმა მსგავსია this საკვანძო სიტყვის გამოყენების, იმ განსხვავებით, რომ super საკვანძო სიტყვა ყოველთვის მიმართავს იმ ქვეკლასის სუპერ კლასს, რომელშიც ის გამოიყენება.

super საკვანძო სიტყვის მეორე ზოგად ფორმას შემდეგი სახე აქვს:

super.წევრი;

აქ **წევრის** ქვეშ იგულისხმება მეთოდი ან ეგზემპლარის ცვლადი.

super საკვანძო სიტყვის გამოყენების მეორე ფორმა იმ სიტუაციებში გვხვდება, როდესაც ქვეკლასის წევრების სახელები ფარავენ სუპერ კლასის იმავე სახელის მქონე წევრებს.

ქვემოთ წარმოდგენილია კლასების მარტივი იერარქია, კერძოდ, super საკვანძო სიტყვის გამოყენების მეორე ფორმა, პროგრამის რეალიზაცია და მისი შედეგები.

პროგრამული რეალიზაცია:

```
1   package usesuper;
    class A{
3       int i;
4   }
5   class B extends A{
        int i;
7       B(int a, int b){
8           super.i=a;
9           i=b;
10      }
11      void show(){
12          System.out.println("i სუპერ კლასში=" + super.i);
13          System.out.println("i ქვეკლასში=" + i);
14      }
15  }
16
17  public class UseSuper {
18      public static void main(String[] args) {
19         B subOb=new B(1,2);
20         subOb.show();
21      }
22  }
```

შედეგი:

```
run:
i სუპერ კლასში=1
i ქვეკლასში=2
BUILD SUCCESSFUL (total time: 0 seconds)
```

მართალია, B კლასის i ცვლადი ეგზემპლარის მიერ დაფარულია A სუპერ კლასის i ცვლადი, მაგრამ super საკვანძო სიტყვა საშუალებას გვაძლევს წვდომა ვიქონიოთ სუპერ კლასში განსაზღვრულ i ცვლადზე.

super საკვანძო სიტყვა ასევე შეგვიძლია გამოვიყენოთ ქვეკლასის მიერ დაფარული მეთოდების გამოსაძახებლად.

აქამდე ჩვენ კლასების მარტივ იერარქიას ვიყენებდით, რომელიც მხოლოდ სუპერ კლასისა და ქვეკლასისგან შედგებოდა. თუმცა, შეგვიძლია შევქმნათ იერარქიები, რომლებიც მემკვიდრეობითობის დონეების ნებისმიერ რაოდენობას მოიცავს. როგორც ვიცით, ქვეკლასი შეიძლება სხვა ქვეკლასის სუპერ კლასის როლში წარმოვადგინოთ. C კლასი შეიძლება იყოს B კლასის ქვეკლასი, რომელიც თავის მხრივ, A კლასის ქვეკლასია. მსგავს სიტუაციებში ყოველი ქვეკლასი მემკვიდრეობით იღებს ყველა მისი სუპერ კლასების მახასიათებლებს. ანუ C კლასი მემკვიდრეობით მიიღებს A და B კლასების ყველა მახასიათებელს.

მრავალდონიანი იერარქიის საილუსტრაციოდ განვიხილოთ **მაგალითი 2,** რომელიც სამკუთხედის აგების წესის დაცვით ჰერონის ფორმულის საფუძველზე სამკუთხედის ფართობს ითვლის.

პროგრამული რეალიზაცია:

```
1  package heroni;
2  class student{
3      double a;
4      student(double a){
5          this.a=a; }}
6  class student1 extends student{
7      double b;
8      student1(double a, double b){
9      super(a);
10     this.b=b;}}
11 class student2 extends student1{
12     double c;
13     student2(double a, double b, double c){
14      super(a,b);
15      this.c=c;}
16      void method1(){
17         if(a+b>c && a+c>b && b+c>a){
18             double p=(a+b+c)/2;
19             double s=Math.sqrt(p*(p-a)*(p-b)*(p-c));
20              System.out.println("ფართობი=" + s);}
21      else
22              System.out.println("დარღვეულია სამკუთხედის აგების წესი.");
23      }}
24     public class Heroni {
25     public static void main(String[] args) {
26         double a=2, b=3, c=4;
27         student2 ob1=new student2(a, b, c);
28         System.out.println("a=" + a);
29         System.out.println("b=" + b);
30         System.out.println("c=" + c);
31         ob1.method1();}}
```

შედეგი:

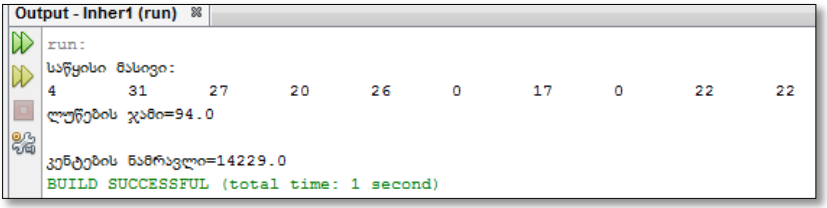
```
Output - Heroni (run)
run:
a=2.0
b=3.0
c=4.0
ფართობი=2.9047375096555625
```

მაგალითი 3. შევადგინოთ პროგრამა, რომელიც 10-ელემენტიან მთელრიცხვა მასივში გამოთვლის მასივის ლუწი მნიშვნელობის ელემენტების ჯამს და კენტი მნიშვნელობის ელემენტების ნამრავლს. ამასთან, მიმართავს კლასების იერარქიულ სტრუქტურას და სუპერ კლასების კონსტრუქტორების გამოძახებას super() მეთოდით განახორციელებს.

პროგრამული რეალიზაცია:

```
1   package inher1;
2   import java.util.Random;
    class student{
4       int a[]=new int[10];
5       student(){
6           Random rand=new Random();
7           for(int i=0; i<a.length; i++){
8               a[i]=(int)(35*rand.nextDouble());
9               System.out.print(a[i] + "\t");}}}
        class student1 extends student{
11          double s;
12          student1(double s){
13              super();
14              this.s=s;}}
15       class student2 extends student1{
16           double p;
17           student2(double s, double p){
18               super(s);
19               this.p=p;}
20           void method1(){
21           for(int i=0; i<a.length; i++)
22               if(a[i]%2==0)
23                   s+=a[i];
24               System.out.println("\nლუწების ჯამი=" + s);}
25           void method2(){
26           for(int i=0; i<a.length; i++)
27               if(a[i]%2==1)
28                   p*=a[i];
29            System.out.println("\nკენტების ნამრავლი=" + p);}}
30          public class Inher1 {
31      public static void main(String[] args) {
32         double s=0, p=1;
33          System.out.println("საწყისი მასივი:");
34          student2 ob1=new student2(s, p);
35          ob1.method1(); ob1.method2();}}
```

შედეგი:



დავალება

1. შეადგინეთ პროგრამა, რომელიც კლასების იერარქიულ სტრუქტურაში სუპერ კლასების კონსტრუქტორების გამოძახების მიზნით super() მეთოდს გამოიყენებს და 15 ელემენტიან მთელრიცხვა მასივში გამოთვლის მაქსიმალური და მინიმალური მნიშვნელობის ელემენტების ნამრავლს.

2. შეადგინეთ პროგრამა, რომელიც კლასების იერარქიულ სტრუქტურაში სუპერ კლასების კონსტრუქტორების გამოძახების მიზნით super() მეთოდს გამოიყენებს და 12 ელემენტიან მთელრიცხვა მასივში გამოთვლის ლუწი მნიშვნელობის ელემენტების საშუალო არითმეტიკულს და კენტი მნიშვნელობის ელემენტების კვადრატების ჯამს.

3. შეადგინეთ პროგრამა, რომელიც კლასების იერარქიულ სტრუქტურაში სუპერ კლასების კონსტრუქტორების გამოძახების მიზნით super() მეთოდს გამოიყენებს და 3x4 განზომილებიან მატრიცაში გამოთვლის კენტი მნიშვნელობის მქონე სვეტებში განთავსებული ელემენტების ნახევარ-ჯამს და ლუწი მნიშვნელობის მქონე სტრიქონებში განთავსებული ელემენტების ნამრავლს.

4. შეადგინეთ პროგრამა, რომელიც კლასების იერარქიულ სტრუქტურაში სუპერ კლასების კონსტრუქტორების გამოძახების მიზნით super() მეთოდს გამოიყენებს და 5x4 განზომილებიან მატრიცაში გამოთვლის ლუწი მნიშვნელობის მქონე სტრიქონებში განთავსებული ელემენტების ჯამიდან კუბური ფესვის მნიშვნელობას და კენტი მნიშვნელობის მქონე სვეტებში არსებული ელემენტების საშუალო არითმეტიკულს.

5. შეადგინეთ პროგრამა, რომელიც კლასების იერარქიულ სტრუქტურაში სუპერ კლასების კონსტრუქტორების გამოძახების მიზნით super() მეთოდს გამოიყენებს და 4x4 განზომილებიან მატრიცაში ცალ-ცალკე გამოთვლის მთავარ და არამთავარ დიაგონალებზე არსებული ელემენტების ნამრავლებს.

ლაბორატორიული სამუშაო №14

მეთოდების ხელახალი განსაზღვრა

როდესაც კლასების იერარქიაში ქვეკლასის მეთოდის სახელი და ტიპის სიგნატურა ემთხვევა სუპერ კლასის მეთოდის ატრიბუტებს, ამბობენ, რომ ქვეკლასის მეთოდი **ხელახლა განსაზღვრავს** სუპერ კლასის მეთოდს.

როდესაც ხელახლა განსაზღვრული მეთოდი გამოიძახება თავისი ქვეკლასიდან, ის ყოველთვის მიმართავს ქვეკლასის მიერ განსაზღვრულ მეთოდის ვერსიას. სუპერ კლასის მიერ განსაზღვრული მეთოდი კი იფარება.

ზემოაღნიშნულის საილუსტრაციოდ განვიხილოთ **მაგალითი1**, რომელიც 12-ელემენტიან მასივის ელემენტებს ანიჭებს მთელი ტიპის ნებისმიერ მნიშვნელობებს და გამოთვლის მასივის 10-ზე მეტი მნიშვნელობის ელემენტების ნამრავლს სუპერ კლასის method1() მეთოდით და 10-ზე ნაკლები მნიშვნელობის ელემენტების კვადრატების ჯამს ქვეკლასის method1() მეთოდით. ვინაიდან კლასების იერარქიაში ორი ერთი და იმავე სიგნატურის მქონე მეთოდი გვაქვს გამოყენებული, ამიტომ აღნიშნულ მაგალითში ქვეკლასის ობიექტის მიერ მეთოდზე მიმართვისას ხდება მისივე კლასში ხელახლა განსაზღვრული მეთოდის გამოძახება და არა სუპერ კლასის method1() მეთოდის, რამეთუ ეს უკა-

ნასკნელი დაფარულია. შედეგად ვიღებთ მხოლოდ 10-ზე ნაკლები მნიშვნელობის ელემენტების კვადრატების ჯამს.

პროგრამული რეალიზაცია:

```
1   package override1;
2   import java.util.Random;
    class student{
4   int a[]=new int[12];
5   student(){
6       Random rand=new Random();
7       for(int i=0; i<a.length; i++){
8           a[i]=(int)(15*rand.nextDouble());
9           System.out.print(a[i] + "\t");}}
        void method1(){
11          double s=1;
12         for(int i=0; i<a.length; i++)
13             if(a[i]>10)
14                 s*=a[i];
15          System.out.println("\nნამრავლი=" + s);}}
16    class student1 extends student{
17        student1(){
18        super();}
          void method1(){
20          double s=0;
21         for(int i=0; i<a.length; i++)
22             if(a[i]<10)
23                 s+=Math.pow(a[i],2);
24          System.out.println("\nკვადრატების ჯამი=" + s);}}
25
26  public class Override1 {
27      public static void main(String[] args) {
28         student1 ob1=new student1();
29         ob1.method1();}}
```

შედეგი:

```
run:
14      1       11      10      3       7       6       13      11      2       8       5
კვადრატების ჯამი=188.0
```

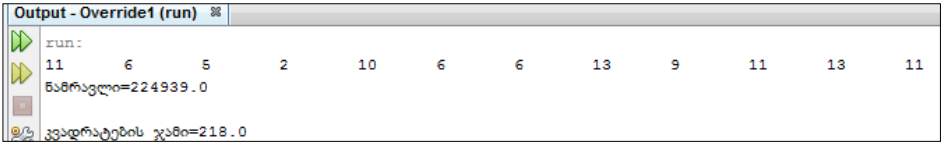
თუ გვინდა წვდომა გვქონდეს სუპერ კლასში განსაზღვრულ მეთოდის ვერსიაზე, საჭიროა მივმართოთ **super** საკვანძო სიტყვას.

აღნიშნულის საილუსტრაციოდ, შეცვლილი სახით წარმოვადგინოთ ზემოთ განხილული პროგრამა. აქ მეთოდი super.method1() იძახებს სუპერ კლასში განსაზღვრულ method1() მეთოდის ვერსიას.

პროგრამის შესრულებაზე გაშვების შედეგად დასმული ამოცანის ამოხსნისას ვიღებთ, როგორც 10-ზე მეტი მნიშვნელობის მქონე ელემენტების ნამრავლს, ისე 10-ზე ნაკლები მნიშვნელობის მქონე ელემენტების კვადრატების ჯამს.

ქვემოთ წარმოდგენილია პროგრამის სახეშეცვლილი (გაუმჯობესებული) ვერსიის შედეგები და შემდგომ პროგრამული რეალიზაცია.

შედეგი:



პროგრამული რეალიზაცია:

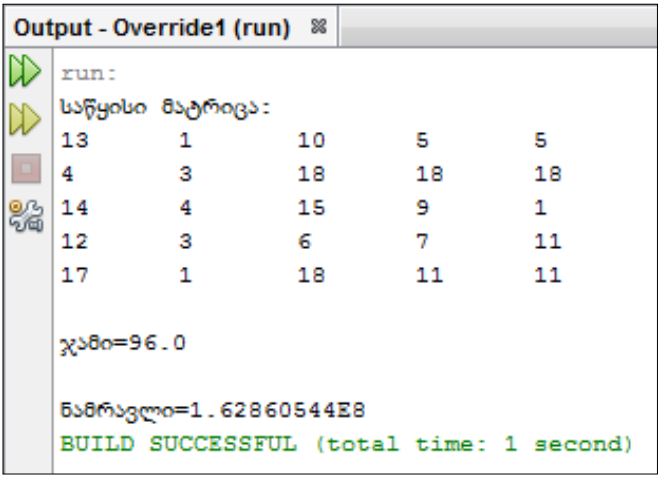
```
1   package override1;
2   import java.util.Random;
    class student{
4   int a[]=new int[12];
5   student(){
6       Random rand=new Random();
7       for(int i=0; i<a.length; i++){
8           a[i]=(int)(15*rand.nextDouble());
9           System.out.print(a[i] + "\t");}}
        void method1(){
11          double s=1;
12          for(int i=0; i<a.length; i++)
13              if(a[i]>10)
14                  s*=a[i];
15          System.out.println("\nნამრავლი=" + s);}}
16    class student1 extends student{
17        student1(){
18        super();}
          void method1(){
20            super.method1();
21          double s=0;
22          for(int i=0; i<a.length; i++)
23              if(a[i]<10)
24                  s+=Math.pow(a[i],2);
25          System.out.println("\nკვადრატების ჯამი=" + s);}}
26
27  public class Override1 {
28      public static void main(String[] args) {
29          student1 ob1=new student1();
30          ob1.method1();}}
```

მეთოდის ხელახალი განსაზღვრა მხოლოდ იმ შემთხვევაში ხდება, როდესაც ორი მეთოდის სახელები და სიგნატურა **იდენტურია.** წინააღმდეგ შემთხვევაში, მეთოდები გადატვირთულს წარმოადგენს.

მაგალითი 2. შევადგინოთ პროგრამა, რომელიც მეთოდის ხელახალი განსაზღვრის გზით 5x5 კვადრატულ მატრიცაში გამოთვლის ჯერ მთავარი დიაგონალის ზემოთ არსებული ელემენტების ჯამს და შემდეგ მთავარი დიაგონალის ქვემოთ არსებული ელემენტების ნამრავლს.

ქვემოთ წარმოდგენილია დასმული ამოცანის ამოხსნის შედეგები და შემდგომ მისი პროგრამული რეალიზაცია.

შედეგი:



პროგრამული რეალიზაცა:

```
1   package override1;
2   import java.util.Random;
    class student{
4   int a[][]=new int[5][5];
5   student(){
6       Random rand=new Random();
7       for(int i=0; i<5; i++){
8         for(int j=0; j<5; j++){
9           a[i][j]=(int)(20*rand.nextDouble());
10          System.out.print(a[i][j] + "\t");}
11          System.out.println();}}
        void method1(){
13          double s=0;
14         for(int i=0; i<5; i++)
15              for(int j=0; j<5; j++)
16              if(i<j) s+=a[i][j];
17          System.out.println("\nჯამი=" + s);}}
18    class student1 extends student{
19        student1(){
20        super();}
          void method1(){
22            super.method1();
23          double p=1;
24          for(int i=0; i<5; i++)
25              for(int j=0; j<5; j++)
26              if(i>j)
27                  p*=a[i][j];
28          System.out.println("\nნამრავლი=" + p);}}
29  public class Override1 {
30      public static void main(String[] args) {
31          System.out.println("საწყისი მატრიცა:");
32         student1 ob1=new student1();
33         ob1.method1();}}
```

დავალება

1. შეადგინეთ პროგრამა, რომელიც მეთოდის ხელახალი განსაზღვრის გზით 6x6 კვადრატულ მატრიცაში შესაბამისი მეთოდის გამოყენებით სტრიქონების მიხედვით განსაზღვრავს ჯერ მაქსიმალური მნიშვნელობის ელემენტებს და შემდეგ იმავე მეთოდის ხელახლა განსაზღვრული ვერსიის გამოყენებით - მინიმალური მნიშვნელობის ელემენტებს.
2. შეადგინეთ პროგრამა, რომელიც მეთოდის ხელახალი განსაზღვრის გზით 4x4 კვადრატულ მატრიცაში შესაბამისი მეთოდის გამოყენებით გამოთვლის ჯერ მატრიცის არამთავარი დიაგონალის ზემოთ განლაგებული ელემენტების საშუალო არითმეტიკულ მნიშვნელობას და შემდეგ იმავე მეთოდის ხელახლა განსაზღვრული ვერსიის გამოყენებით - არამთავარი დიაგონალის ქვემოთ განლაგებული ელემენტების საშუალო არითმეტიკულ მნიშვნელობას.
3. შეადგინეთ პროგრამა, რომელიც მეთოდის ხელახალი განსაზღვრის გზით 10-ელემენტიან მთელრიცხვა მასივში შესაბამისი მეთოდის გამოყენებით გამოთვლის ჯერ ლუწი მნიშვნელობის ელემენტების ნამრავლს და შემდეგ იმავე მეთოდის ხელახლა განსაზღვრული ვერსიის გამოყენებით - კენტი მნიშვნელობის ელემენტების ნამრავლს.

ლაბორატორიული სამუშაო №15

აბსტრაქტული კლასები და მათი გამოყენება

ზოგ შემთხვევაში საჭიროა განისაზღვროს სუპერ–კლასი, რომელიც ყოველი მეთოდის სრული რეალიზაციის წარმოდგენის გარეშე ახდენს გარკვეული აბსტრაქციის სტრუქტურის გამოცხადებას. ანუ, გვიწევს ისეთი სუპერ–კლასის შექმნა, რომელიც განსაზღვრავს მხოლოდ განზოგადებულ ფორმას, რასაც მისი ყველა ქვეკლასი გარკვეული დეტალების დამატებით ერთობლივად გამოიყენებს. ასეთი კლასი განსაზღვრავს იმ მეთოდების არსს, რომელთა რეალიზაცია ქვეკლასებმა უნდა მოახდინოს. მსგავსი სიტუაცია იქმნება მაშინ, როდესაც სუპერ კლასს არ ძალუძს მეთოდის სრულყოფილი რეალიზაციის შემუშავება. იმისათვის, რომ მოვითხოვოთ, რომ გარკვეული მეთოდები ხელახლა განისაზღვროს ქვეკლასის მიერ, შეგვიძლია **abstract** ტიპის მოდიფიკატორი გამოვიყენოთ. ზოგჯერ ასეთ მეთოდებს ქვეკლასის კომპეტენციაში მყოფს უწოდებენ, რადგან სუპერ კლასში მათი არანაირი რეალიზაცია გათვალისწინებული არ არის.

ამგვარად, ქვეკლასმა ხელახლა უნდა განსაზღვროს ეს მეთოდები, რადგან სუპერ კლასში განსაზღვრული ამ მეთოდების ვერსიის გამოყენების უფლება მას არ აქვს. აბსტრაქტული მეთოდის გამოცხადების ზოგადი ფორმა შემდეგია:

abstract ტიპი სახელი(პარამეტრების_სია);

როგორც ხედავთ, აღნიშნული ფორმა მეთოდის ტანს არ მოიცავს. ნებისმიერი კლასი, რომელიც ერთ ან რამდენიმე აბსტრაქტულ მეთოდს შეიცავს, ასევე **აბსტრაქტულ კლასად** უნდა იყოს გამოცხადებული. ამისათვის საკმარისია კლასის გამოცხადებისას class საკვანძო სიტყვას წინ უსწრებდეს **abstract** მოდიფიკატორი. აბსტრაქტული კლასი არ შეიძლება შეიცავდეს ობიექტებს. ანუ, აბსტრაქტული კლასის უშუალო კონკრეტიზაცია new ოპერატორით დაუშვებელია. ასეთი ობიექტები უსარგებლო იქნებოდა, რადგან აბსტრაქტული კლასი სრულყოფილად არ არის განსაზღვრული. ასევე, დაუშვებელია აბსტრაქტული კონსტრუქტორებისა და აბსტრაქტული სტატიკური მეთოდების გამოცხადება. აბსტრაქტული კლასის ნებისმიერმა ქვეკლასმა ან უნდა მოახდინოს თავისი აბსტრაქტული სუპერ კლასის ყველა მეთოდის რეალიზება, ან თავადაც უნდა გამოცხადდეს, როგორც აბსტრაქტული კლასი.

მაგალითი 1. შევადგინოთ პროგრამა, რომელიც აბსტრაქტული კლასის რეალიზების გზით ერთიდან *n*-მდე ყველა წესიერ რიცხვს (ანუ რიცხვს, რომლის გამყოფების ჯამი ამავე რიცხვის ტოლია) განსაზღვრავს.

პროგრამული რეალიზაცია:

```
1   package abstractdemo;
    abstract class student{
        abstract void method1();
4   }
5   class student1 extends student{
        void method1(){
7           for (int n=1; n<10000; n++){
8            int sum = 0, k = 1;
9            while(k < n){
10           if(n % k == 0) sum += k;
11           k++;}
12       if (n == sum) System.out.println (" n = " + n);}}}
13  public class AbstractDemo {
14      public static void main(String[] args) {
15        student1 ob1=new student1();
16        ob1.method1();}}
```

შედეგი:

```
Output - AbstractDemo (run)
run:
 n = 6
 n = 28
 n = 496
 n = 8128
BUILD SUCCESSFUL (total time: 0 seconds)
```

მაგალითი 2. შევადგინოთ პროგრამა, რომელიც აბსტრაქტული კლასის რეალიზების გზით ნებისმიერი მთელი დადებითი რიცხვის ყველა გამყოფს განსაზღვრავს.

პროგრამული რეალიზაცია:

```
1   package abstractdemo;
    abstract class student{
        abstract void method1();
4   }
5   class student1 extends student{
        void method1(){
7           int num=1024;
8           int half=num/2;
9           int div=2;
10          while(div<=half){
11              if(num%div==0)
12                  System.out.print(div +"\t");
13              div++;}}}
14
15      public class AbstractDemo {
16      public static void main(String[] args) {
17        student1 ob1=new student1();
18        ob1.method1();
19        System.out.println ();}}
```

შედეგი:

```
Output - AbstractDemo (run)
run:
2       4       8       16      32      64      128     256     512
```

მართალია, აბსტრაქტული კლასები ობიექტების კონკრეტიზაციისთვის არ შეიძლება გამოვიყენოთ, მაგრამ შეგვიძლია გამოვიყენოთ ობიექტებზე მიმართვის შესაქმნელად.

ამ თვისების გამოყენების ნიმუში შემდეგ პროგრამაშია ნაჩვენები. მასში area() მეთოდი განსაზღვრულია, როგორც Figure კლასის აბსტრაქტული მეთოდი, რომლის ხელახალ განსაზღვრას Figure კლასისგან წარმოებული ქვეკლასები (ყველა ქვეკლასი) ახდენს.

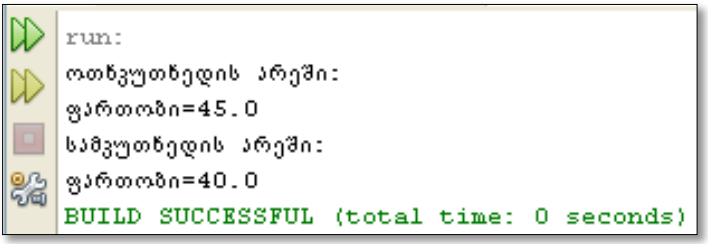
როგორც main() მეთოდის კომენტარიდან ჩანს, Figure ტიპის ობიექტების გამოცხადება უკვე დაუშვებელია, რამეთუ ეს კლასი აბსტრაქტულია. Figure კლასის ყველა ქვეკლასი აბსტრაქტული area() მეთოდის ხელახალ განსაზღვრას ახდენს. თუ შევქმნით ქვეკლასს, რომელიც area() მეთოდს ხელახლა არ განსაზღვრავს, კომპილაციის დროს მივიღებთ შეტყობინებას შეცდომის შესახებ.

მართალია, Figure ტიპის ობიექტის შექმნა დაუშვებელია, მაგრამ უფლება გვაქვს შევქმნათ Figure ტიპის მიმართვითი ცვლადი. აქ figref ცვლადი გამოცხადებულია, როგორც მიმართვა Figure ტიპზე და ამდენად, მისი გამოყენება შეგვიძლია Figure კლასისგან წარმოებული ნებისმიერი ქვეკლასის ობიექტზე მიმართვისათვის.

პროგრამული რეალიზაცია:

```
1   package findareas;
    abstract class Figure{
3       double dim1;
4       double dim2;
5       Figure(double a, double b){
6           dim1=a;
7           dim2=b;
8       }
        abstract double area();
10    }
11  class Rectangle extends Figure{
12      Rectangle(double a, double b){
13          super(a,b);
14      }
        double area(){
16          System.out.println("ოთხკუთხედის არეში:");
17          return dim1 * dim2;
18      }
19  }
20  class Triangle extends Figure{
21      Triangle(double a, double b){
22          super(a,b);
23      }
        double area(){
25          System.out.println("სამკუთხედის არეში:");
26          return dim1 * dim2/2;
27      }
28  }
29  public class FindAreas {
30      public static void main(String[] args) {
31       // Figure f=new Figure(10, 10);  ახლა უკვე დაუშვებელია!!!
32        Rectangle r=new Rectangle(9, 5);
33        Triangle t= new Triangle(10, 8);
34        Figure figref;
35        figref=r;
36          System.out.println("ფართობი=" + figref.area());
37         figref=t;
38          System.out.println("ფართობი=" + figref.area());
39      }
40  }
```

შედეგი:



<u>დავალება</u>

1. შეადგინეთ პროგრამა, რომელიც აბსტრაქტული კლასის რეალიზების გზით გამრავლების ტაბულას კონსოლზე გამოიტანს.
2. შეადგინეთ პროგრამა, რომელიც აბსტრაქტული კლასის რეალიზების გზით ერთიდან 1000-მდე ყველა მარტივ რიცხვს კონსოლზე გამოიტანს.
3. შეადგინეთ პროგრამა, რომელიც აბსტრაქტული კლასის რეალიზების გზით ორი ნატურალური რიცხვის უმცირესი საერთო ჯერადის მნიშვნელობას განსაზღვრავს.
4. შეადგინეთ პროგრამა, რომელიც აბსტრაქტული კლასის რეალიზების გზით კონსოლზე შემდეგ სურათს წარმოგვიდგენს.

&&&&&
&&&&
&&&
&&
&

ლიტერატურა

1. Herbert Schildt, *The Complete Reference, Java Seventh Edition,* The McGraw-HIll Companies, Inc. 2007, ISBN 978-0-07-163177-8, 1024 p.
2. H.N. Deitel, Java How to Program, Seveth edition, Prentice Hall, 2006, 1705 p.
3. Ivor Horton, Ivor Horton's Beginning Java™ 2, JDK™ 5 Edition, Published by Wiley Publishing, Inc., 2008, ISBN 0-7645-6874-4, 1501 p.
4. http://java.sun.com/docs/books/tutorial/index.html

სარჩევი

გვ.

იბეჭდება ავტორთა მიერ წარმოდგენილი სახით

რედაქტორი: გოჩა დალაქიშვილი

გადაეცა წარმოებას 19.09.2014ხელმოწერილია დასაბეჭდად 17.09.2014 ქაღალდის ზომა 60X84. პირობითი ნაბეჭდი თაბახი 8. ტირაჟი 100 ეგზ.

ი.მ. ‘‘გოჩა დალაქიშვილი’’, ვარკეთილი 3, კ. 333, ბ. 38