

**Александар Картељ
Владимир Филиповић
Душан Тошић**

ОБЈЕКТНО ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ

ПРОГРАМСКИ ЈЕЗИК ЈАВА - 1



Математички факултет
Универзитет у Београду
2023. година

Универзитет у Београду

Математички факултет

Објектно оријентисано програмирање

Програмски језик Јава – 1

Александар Картељ, Владимир Филиповић, Душан Тошић

Београд, 2023.

Садржај

Предговор	xiii
1. Решавање проблема помоћу рачунара	1
1.1. Опис поступка решавања проблема помоћу рачунара	1
1.1.1. Водопадни модел	1
Фаза анализе и дефиниције проблема	3
Фаза дизајна (пројектовања) софтвера	4
Фаза имплементације	4
Фаза тестирања софтвера	4
Фаза коришћења и подршке	5
1.1.2. Спирални модел развоја софтвера	5
1.2. Језички процесори	8
1.3. Резиме	11
1.4. Питања и задаци	12
2. Објектно оријентисано програмирање	13
2.1. Карактеристике објектно оријентисаног програмирања	13
2.2. Историјат и развој објектно оријентисаног програмирања	14
2.3. Основни појмови објектно оријентисаног програмирања	14
2.3.1. Класе и инстанце	16
2.3.2. Наслеђивање и композиција	18
2.3.3. Касно везивање	21
2.3.4. Учауривање	21
2.4. Предности и мане објектно оријентисаног програмирања	25
2.5. Резиме	26
2.6. Питања и задаци	27
3. Неке програмске парадигме	29

3.1. Императивно програмирање	29
3.1.1. Структурно програмирање	30
3.1.2. Модуларно програмирање	32
3.2. Декларативно програмирање	33
3.2.1. Логичко програмирање	33
3.2.2. Функционално програмирање	35
3.2.3. Објектно оријентисано програмирање	36
3.3. Резиме	38
3.4. Питања и задаци	39
4. Карактеристике програмског језика Јава	41
4.1. Историјат и развој програмског језика и окружења Јава	41
Почетак и рани развој	41
Јава и Oracle	44
4.2. Постављени циљеви приликом развоја програмског језика и окружења Јава	46
4.3. Типови Јава апликација	48
4.3.1. Десктоп апликације са графичким корисничким интерфејсом	48
4.3.2. Апликације које се покрећу из команде линије	49
4.3.3. Апликације за мобилне уређаје	49
4.3.4. Аплети (веб апликације на клијентској страни)	50
4.3.5. Сервлети и Јава серверске стране	50
4.3.6. Веб сервиси	51
4.3.7. Библиотеке класа	51
4.4. Процес превођења и извршавања Јава програма	51
ЈИТ Јава преводацац	54
4.4.1. Јава виртуелна машина	56
Архитектура JVM	57
Меморија JVM	57
Скупљач отпадака	59

4.5. Јава алати за развој – JDK	61
4.5.1. Библиотеке класа, пакети и модули	62
Јава API	62
Модули	63
4.5.2. Структура JDK директоријума након инсталације	64
4.5.3. Издања Јава окружења (стандардно и пословно)	67
Централни Јава API	67
Додатни Јава API (Enterprise, Server, Beans, итд.)	69
Java Enterprise API	69
Java Server API	70
Java Security API	71
4.7. Резиме	71
4.8. Питања и задаци	72
5. Језици и опис конструкција језика Јава	73
5.1. Граматика, синтакса и семантика	73
5.1.1. Бекусова нотација	75
5.2. Елементарне конструкције језика Јава	77
5.2.1. Идентификатори	77
5.2.2. Кључне речи	78
5.2.3. Литерали	79
Целобројни литерали	79
Реални литерали	80
Логички литерали	81
Знаковни литерали	81
Литерали-ниске	82
5.2.4. Сепаратори	83
5.2.5. Оператори и изрази	83
Аритметички оператори и изрази	84
Битовни оператори и изрази	85
Релациони оператори и изрази	86

Логички оператори и изрази	87
Условни оператор и израз	88
Инстанцни оператор и израз	88
Оператор и израз за прављење објекта	89
Оператори доделе и изрази доделе	89
Арност, асоцијативност и приоритет оператора	90
5.2.6. Белине	91
5.2.7. Коментари	92
5.3. Типови података у Јави	93
5.3.1. Примитивни типови података	94
Целобројни тип података	95
Реални тип података	96
Знаковни тип података	97
Логички тип података	98
5.3.2. Објектни тип	98
Објекти и класе	99
5.3.3. Експлицитна конверзија типа	100
5.4. Променљиве	101
5.4.1. Декларација и иницијализација вредности променљиве	102
5.5. Наредбе	105
5.5.1. Наредба изрази	106
5.5.2. Блок	106
5.5.3. Наредбе гранања	108
Наредба if	108
Непотпуна наредба if	108
Потпуна наредба if	109
Наредба switch и наредба break	114
5.5.4. Наредбе понављања	118
Наредба while	118
Наредба do-while	120

Наредба for (бројачки циклус)	123
Наредба break и циклуси	129
Наредба continue и циклуси	130
5.5.5. Обележена наредба	131
5.5.6. Празна наредба	133
5.6. Резиме	134
5.7. Питања и задаци	134
6. Коришћење класа и објеката испоручених уз JDK	139
6.1. Приступ систему, класа System	139
6.1.1. Приказ текста	139
6.1.2. Мерење протеклог времена	141
6.1.3. Захтев за покретањем скупљача отпадака, метод System.gc()	143
6.1.4. Излазак из апликације, метод System.exit()	146
6.2. Рад са нискама (инстанцама класе String)	147
Карактеристике ниски, непроменљивост	148
Прављење ниски и неке стандардне методе над ниском	148
Поређење ниски	151
Класа StringBuilder	152
6.3. Рад са омотачима података примитивног типа	154
Обмотавање примитивних типова	154
Одмотавање омотач типова	155
6.4. Рад са парсерима, инстанцама класе java.util.Scanner	157
Парсирање података из ниске	157
Парсирање података са стандардног улаза	159
6.5. Рад са математичким функцијама, класа Math	160
6.6. Рад са датумима и временима	164
6.7. Рад са псеудослучајним бројевима	170
6.8. Резиме	174
6.9. Питања и задаци	174

7. Низови у Јави	177
7.1. Декларација и иницијализација низа	179
7.2. Низовна променљива и индексна променљива	181
7.3. Низови и циклуси	182
7.4. Аргументи команде линије код улазне тачке програма	190
7.5. Вишедимензионални низ	192
7.5.1. Дводимензионални низ	192
7.5.2. Тродимензионални низ и низови већих димензија	198
7.6. Коришћење класе Arrays	200
Сортирање низа	200
Бинарна претрага над низом	201
7.7. Методи са аргументима променљиве дужине	205
7.8. Резиме	208
7.9. Питања и задаци	208
8. Класе, пакети, поља, методи и објекти у Јави	211
8.1. Класе у Јави	211
8.1.1. Објекат и референца на објекат	213
8.1.2. Прављење објекта – инстанце дате класе	213
8.1.3. Објекти класе Object	215
8.1.4. Поређење референци на објекат	216
8.2. Организација класа по пакетима	218
8.2.1. Дефинисање пакета, наредба package	219
8.2.2. Увоз класа из пакета, наредба import	220
8.3. Класе и објекти – поља	222
8.3.1. Дефиниција поља	222
8.3.2. Приступ пољу у примерку дате класе	223
8.3.3. Статичка (класна) поља	224
8.3.4. Опсег важења за променљиве и поља	226
8.4. Класе и објекти – методи	227
8.4.1. Дефиниција и позив метода	227

8.4.2. Позив метода и кључна реч <code>this</code>	229
8.4.3. Преоптерећење метода	232
8.4.4. Статички (класни) методи	235
8.5. Класе – наслеђивање	236
8.5.1. Приступање пољима наткласа и њихово сакривање	238
Приступ пољима наткласе у методима, кључна реч <code>super</code>	238
Сакривање поља	240
8.5.2. Испитивање да ли објекат припада класи	240
8.5.3. Конверзија између објеката	241
8.5.4. Позивање метода наткласа и њихово превазилажење	243
Позивање методе наткласе, кључна реч <code>super</code>	243
Превазилажење метода	244
Превазилажење методе за испис објекта – <code>toString()</code>	246
Превазилажење методе за поређење једнакости објеката – <code>equals()</code>	247
Превазилажење метода за хеш-код објекта	248
8.5.5. Полиморфизам	252
8.6. Подешавање почетног стања објекта	254
8.6.1. Иницијализациони блок	255
8.6.2. Конструктор	256
Преоптерећење конструктора и употреба референце <code>this</code>	258
Референцијална зависност објеката и копирајући конструктор	259
Позив конструктора наткласе, референца <code>super</code>	265
8.7. Модификатори видљивости	267
8.7.1. Модификатор <code>public</code>	268
8.7.2. Модификатор <code>package</code>	271
8.7.3. Модификатор <code>protected</code>	272
8.7.4. Модификатор <code>private</code>	274
Заштита поља применом модификатора <code>private</code>	276
8.8. Модификатор ограничавања – <code>final</code>	280

Спречавање наслеђивања	280
Константна поља	281
Спречавање превазилажења метода	281
8.9. Неке динамичке структуре података	282
8.9.1. Саморастући низ	282
8.9.2. Повезана листа	286
8.10. Резиме	293
8.11. Питања и задаци	294
9. Напредни рад са класама и објектима	297
9.1. Апстрактне класе	297
9.1.1. Дефинисање апстрактне класе	298
9.1.2. Наслеђивање између апстрактних и конкретних класа	300
9.2. Интерфејси	316
9.2.1. Дефинисање интерфејса	317
9.2.2. Имплементирање интерфејса од стране класе	318
9.2.3. Вишеструко наслеђивање и интерфејси	320
9.2.4. Проширивање интерфејса	325
9.2.5. Параметри типа интерфејса	338
9.3. Интерфејси у JDK-у	344
9.3.1. Сортирање, интерфејс Comparable	344
9.3.2. Вишекритеријумско сортирање, интерфејс Comparator	349
9.3.3. Клонирање објеката, интерфејс Cloneable	356
9.4. Принципи и препоруке објектно оријентисаног дизајна	368
9.4.1. SOLID принципи	369
Принцип појединачне одговорности	370
Принцип отворености и затворености	375
Принцип заменљивости	380
Принцип раздвајања интерфејса	383
Принцип инверзије зависности	389
9.4.2. Објектно оријентисани дизајн – препоруке	396

Користити наслеђивање искључиво за моделирање односа „јесте“	396
Заједничке операције и поља сместити у наткласе	399
Не користити наслеђивање, сем уколико оно има смисла за све методе класе из које се наслеђује	400
Приликом превазилажења метода не мењати очекивано понашање	400
Дати предност композицији и садржавању у односу на наслеђивање	401
Избегавати употребу заштићених поља	401
Користити полиморфизам, а не информације о типу	401
9.5. Резиме	402
9.6. Питања и задаци	402
10. Угнежђене класе	405
10.1. Статичке угнежђене класе	406
10.2. Унутрашње класе	410
10.2.1. Локалне унутрашње класе	416
10.2.2. Анонимне класе	419
10.3. Резиме	421
10.4. Питања и задаци	422
11. Изузеци и тврдње	423
11.1. Изузеци	423
11.1.1. Класе изузетака	425
Изузеци класе Error	425
Изузеци класе RuntimeException	427
Изузеци класе Exception	429
11.1.2. Руковање изузецима	430
Блок try	432
Блок catch	433
Вишеструки catch блок	433
Блок finally	434

11.1.3. Прослеђивање (пропагирање) изузетака	436
11.1.4. Избацивање изузетака	438
11.1.5. Препоруке за рад са изузецима	440
11.2. Тврдње	441
11.2.1. Наредба assert	442
11.2.2. Препоруке за рад са тврдњама	445
11.3. Резиме	445
11.4. Питања и задаци	445
12. Набројиви (енумерисани) тип	447
12.1. Набројиви типови и наредба switch	449
12.2. Обогаћивање набројивих типова, конструктори и методи	451
12.3. Реализација набројивог типа помоћу класе	456
12.4. Резиме	458
12.5. Питања и задаци	458
13. Генерички тип	461
13.1. Негенерички тип	461
13.2. Појам, дефинисање и предности генеричког типа	463
13.2.1. Генерички интерфејси и њихова имплементација	466
13.3. Самостални генерички метод	474
13.4. Ограничења за типове	475
13.5. Генерици и виртуелна машина	479
13.6. Генерици и наслеђивање	482
13.7. Резиме	483
13.8. Питања и задаци	484
14. Колекције и речници	487
14.1. Интерфејс и имплементација	488
14.2. Колекције и итератори	494
14.2.1. Интерфејс Collection	494
14.2.2. Интерфејси Iterable и Iterator	495
14.2.3. Операције над колекцијом коришћењем итератора	497

14.3. Колекцијски интерфејси	498
14.3.1. Листа, интерфејс List	499
14.3.2. Ред, интерфејси Queue и Dequeue	500
14.3.3. Скуп, интерфејс Set	501
14.4. Колекцијске класе	502
14.4.1. Листе	503
Двоструко повезана листа, класа LinkedList	503
Низовна листа, класа ArrayList	506
14.4.2. Скупови	511
Хеш-скуп, класа HashSet	511
Дрво-скуп, класа TreeSet	517
Остале имплементације скупова	520
14.4.3. Редови	524
Низовни ред са два краја, класа ArrayDeque	524
Ред са приоритетом, класа PriorityQueue	526
14.5. Речници	528
14.5.1. Интерфејс Map	528
14.5.2. Класе за речнике	530
Хеш-речник, класа HashMap	531
Дрво-речник, класа TreeMap	533
14.6. Генерици и колекције	534
14.6.1. Џокер тип	536
14.6.2. Генерички колекцијски методи имплементирани у JDK	540
Сортирање колекције	540
Мешање колекције	541
Бинарна претрага	541
Преглед значајнијих метода услужне класе Collections	542
14.7. Апстрактне класе као основа за колекције	543
14.8. Резиме	552
14.9. Питања и задаци	552

15. Улаз и излаз	555
15.1. Блокирајући улаз и излаз – java.io	556
15.1.1. Улазни и излазни токови података	557
Улазни ток података, InputStream	558
Излазни ток података, OutputStream	561
15.1.2. Читачи и писачи	566
Читачи	566
Писачи	569
15.1.3. Уланчавање токова	571
15.1.4. Рад са датотекама – класа File	573
15.2. Парсирање приликом читања/уписа – класа Scanner	578
15.3. Резиме	580
15.4. Питања и задаци	580
Додатак А – Инсталација Јаве и развојног окружења	583
Инсталација JDK	583
Инсталација развојног окружења Eclipse под Windows ОС	585
Инсталација развојног окружења IntelliJ под Windows ОС	589
Додатак Б – Упутство за употребу GitHub репозиторијума и подешавање ћирилице	593
Преузимање (клонирање) GitHub репозиторијума	593
Уграђивање GitHub материјала у постојећи Eclipse пројекат	595
Уграђивање GitHub материјала у постојећи IntelliJ пројекат	599
Литература	601

Предговор

Ова књига је, првенствено, намењена студентима Математичког факултета у Београду и требало би да послужи као уџбеник за предмет Објектно оријентисано програмирање. Међутим, књига може бити од користи свакоме ко жели да научи програмски језик Јава и да се упозна са принципима програмирања. У књизи је покривена верзија Јава 17LTS.

На спском говорном подручју постоји велики број књига које се односе на програмски језик Јава и пратеће технологије. Ова књига је специфична због повезаности са предметом Објектно оријентисано програмирање и усклађена је са програмом овог предмета. Књига представља свеобухватни водич за програмски језик Јава, али не и за све његове пратеће библиотеке. Опис коришћења свих кључних библиотека језика Јава превазилази обим ове књиге. За боље разумевање пожељно је да читалац познаје програмски језик С, али није неопходно.

За опис конструкција Јава језика коришћена је Бекусова нотација. Скоро сваки формални опис синтаксе Јава-конструкција пропраћен је примерима. Додатно, већина поглавља садржи већи број комплетно урађених задатака. Кроз решења задатака приказују се најважније могућности језика Јава и принципи програмирања. За највећи број решења наведени су тест примери да би се видело како је организован улаз и у каквом облику се добијају излазни подаци. Решени примери из уџбеника доступни су и на GitHub страни посвећеној уџбенику <https://github.com/matf-oop-java/tom-1>.

Књига се састоји од 15 поглавља, два додатка и литературе. На крају сваког поглавља су наведена питања и задаци за вежбање. Ако неко учи било који програмски језик, па самим тим и програмски језик Јава, пожељно је да самостално уради што већи број задатака за вежбање.

Аутори су настојали да свуда, где је то могуће, користе ћирилично писмо. Коментари у програмима и излазни резултати, који садрже текст, исписани су ћирилицом. Када су у питању имена програмских технологија, она су скоро искључиво писана у оригиналу, тј. без транскрипције са енглеског језика. Изузетак су назив програмског језика Јава, реч квиксорт (енг. quick sort) и још неколико других речи. Оригинални запис речи Јава је задржан у ситуацијама када реч учествује

у формирању назива неке специфичне Јава технологије, на пример, Java Security API или Java FX.

С обзиром на то да је ово прво издање књиге, грешке су могуће. Штампарске грешке у рукопису су скоро неизбежне, иако су текст пажљиво прочитали аутори и већи број колега. Међу њима велику захвалност дугујемо др Милани Грбић, која је, поред пажљивог читања, помогла и у конципирању питања и задатака на крајевима поглавља и др Сташи Вујичић-Станковић, која је пажљиво прочитала сва поглавља књиге и дала врло детаљне и корисне коментаре. Поред њих, захваљујемо се проф. др Драгану Матићу на уочавању неких терминолошких и суштинских грешака, проф. др Филипу Марићу на предложеној реорганизацији примера у почетним поглављима књиге и проф. др Предрагу Јаничићу на давању смерница за припрему књиге у складу са правилима издавача. Захваљујемо се и нашим студентима који су помогли у отклањању словних грешака и логичких недоследности. Реч је о студентима: Андреа Андрејевић, Ирина Шевић, Стефан Миленковић, Лука Петковић и Урош Динић. Унапред се извињавамо онима које смо евентуално заборавили да споменемо.

Посебну захвалност дугујемо рецензентима проф. др Ненаду Митићу и проф. др Вељку Милутиновићу. Они су пажљиво прочитали цео рукопис и корисним сугестијама допринели да ова књига буде квалитетнија.

Аутори

6. Коришћење класа и објеката испоручених уз JDK

У овом поглављу ће бити представљене неке од основних могућности Јава библиотеке класа попут исписа и читања са стандардног излаза/улаза, мерења протеклог времена, рада са текстом, псеудослучајним бројевима, математичким функцијама итд.

6.1. Приступ систему, класа System

Класа `System` се може назвати услужном класом (енг. utility class) будући да се не очекује прављење њених објеката. Акцент је на позивању њених метода у тзв. статичкој нотацији, која је коришћена и у претходним поглављима. Статичка поља и методи ће бити детаљније разматрани у секцијама [8.3.3](#) и [8.4.4](#). Засад је довољно рећи да су у питању ентитети који нису зависни од постојања објекта посматране класе, тј. може им се приступати коришћењем имена класе на следећи начин (што доста подсећа на стил процедуралног програмирања):

```
NazivKlase.statickiMetod()
NazivKlase.statickoPolje
```

6.1.1. Приказ текста

Класа `System` има статичко поље `System.out`, које представља објекат везан за стандардни излаз (попут `stdout` у језику C). `System.out` подразумевано показује на конзолу па се запис текста манифестује исписом на конзоли.

Овај објекат поседује методе (нестатичке) којима се приступа употребом тачка нотације. Следи преглед неких, често коришћених, метода.

```
System.out.print(tip arg)
System.out.println(tip arg)
System.out.printf(String format, Object...args)
```

`System.out.print()` исписује аргумент на стандардни излаз (аргумент је произвољног типа, било примитивног или објектног).

`System.out.println()` ради исто што и `System.out.print()` и уз то додаје ознаку за прелазак у нови ред.

`System.out.printf()` испишује текст форматирано, попут функција за форматирани испис у C-у (`printf()`, `sprintf()`, `fprintf()`).

Написати Јава програм који приказује употребу форматираних и неформатираних исписа различитих типова података на стандардном излазу.

```
public class RazlicitiIspisi {
    public static void main(String[] args) {
        System.out.print("Пример текста");
        System.out.print(" са конкатенацијом и бројем "+4);
        System.out.println(); // испис празног реда
        System.out.println(67.4);
        double x = 26.43462;
        int y = 43243;
        float z = 1645.14f;
        System.out.printf("x=%9.6f y=%8d z=%.3f", x, y, z);
        System.out.println();
        String s = "Неки текст";
        char c = 'c';
        System.out.printf("Стринг се умеће форматом %s %s", s);
        System.out.printf(" док се карактер умеће форматом %%%c%s", c,
            System.lineSeparator());
        System.out.printf("%s\t%s\n", "KPAJ", "ISPISA");
    }
}
```

Када се користи форматирани испис, први број после знака `%` представља број места која ће се користити за комплетан запис броја. Ако нема овог броја, број места се аутоматски прилагођава дужини записа броја. Број после тачке означава колико места се користи за разломљени део броја. На пример, `%9.6f` значи да ће бити употребљено девет места, од чега ће 6 бити употребљено за запис разломљеног дела реалног броја.

Будући да знак `%` има специјално значење код форматираних исписа, да би се записао знак `%` потребно је записати га два пута у оквиру ниске за запис формата (`%%`).

`System.lineSeparator()` враћа секвенцу за крај реда прилагођену оперативном систему, али се ово може краће записати као `%n` (у оквиру форматираног исписа).

Приметити да је приликом исписа могуће комбиновати различита писма – ово је последица тога што су ниске у Јави конципиране као секвенце Unicode карактера.

Следи резултат извршавања програма.

Пример текста са конкатенацијом и бројем 4

67.4

x=26.434620 y= 43243 z=1645.140

Стринг се умеће форматом %s Неки текст док се карактер умеће форматом %c
КРАЈ ISPISA

Док је `System.out` надлежан за испис информација у току нормалног функционисања програма, `System.err` је препоручени начин за испис грешака које се јављају у току извршавања или приликом покретања програма. На пример, ако би се тражило да корисник унесе број, а корисник уместо тога унесе секвенцу карактера „4543-АФД-325-11“, програм би могао да испише на `System.err` упозорење о лошем запису броја и да прекине извршавање или да захтева поновни унос. Списак доступних метода у оквиру `System.err` је идентичан као и у случају `System.out`, што је и очекивано с обзиром да оба представљају ток података под називом `PrintStream` (више о токовима у секцији [15.1](#)).

Локација исписа `System.out` и `System.err` се може променити применом метода `System.setOut()` и `System.setErr()`. На пример, може се направити датотека која ће се користити за испис, уместо исписа на конзолу.

6.1.2. Мерење протеклог времена

Мерење времена, у оквиру програма, може бити корисно у анализи перформанси захтевних делова кода. `System` класа омогућава два начина за мерење времена:

1. употребом метода `System.currentTimeMillis()` и
2. употребом метода `System.nanoTime()`.

Принцип рада оба метода је исти – враћају време протекло од неког референтног тренутка у прошлости (Unix epoch - 1. јануар 1970 по Гриничу), с тим што је други метод прецизнији, јер рачуна протекло време у наносекундама, за разлику од првог који рачуна у милисекундама. На овај начин је могуће индиректно закључити колико траје неки интервал тако што се одузму „апсолутна“ времена догађаја за крај и почетак интервала.

Написао сам Јава програм који мери време извршавања метода за сабирање бројева од 1 до n. Тестирати програм за различите вредности n, као и за различите методе за мерење времена. Упоредити измерене вредности у милисекундама.

```
public class IzmeriVreme {
    public static long sumiraj(int n) {
        long suma = 0;
        for (int i = 0; i < n; i++)
            suma += i;
        return suma;
    }

    public static void main(String args[]) {
        for (int n = 10000000; n <= 1000000000; n *= 10) {
            long pocetak1 = System.nanoTime();
            long suma = sumiraj(n);
            System.out.printf("Сума бројева до %d је %d\n", n, suma);
            long kraj1 = System.nanoTime();
            System.out.println("Време у ns са nanoTime(): "
                + (kraj1 - pocetak1));
            System.out.printf("Време у ms са nanoTime(): %.5g%s",
                (kraj1-pocetak1)/1000000.0, System.lineSeparator());

            long pocetak2 = System.currentTimeMillis();
            suma = sumiraj(n);
            System.out.printf("Сума бројева до %d је %d\n", n, suma);
            long kraj2 = System.currentTimeMillis();
            System.out.println("Време у ms са currentTimeMillis(): "
                + (kraj2 - pocetak2));
            System.out.println("-----");
        }
    }
}
```

Може се приметити да `nanoTime()` даје много прецизнија мерења и да број протеклих милисекунди метода `currentTimeMillis()` постаје тачнији (или мање погрешан) са растом `n`, тј. дужином трајања метода `sumiraj()`.

Следи резултат рада програма.

```
Сума бројева до 10000000 је 49999995000000
Време у ns са nanoTime(): 10856500
Време у ms а nanoTime(): 10.857
Сума бројева до 10000000 је 49999995000000
Време у ms са currentTimeMillis(): 4
-----
Сума бројева до 100000000 је 4999999950000000
Време у ns са nanoTime(): 22943200
Време у ms са nanoTime(): 22.943
Сума бројева до 100000000 је 4999999950000000
Време у ms са currentTimeMillis(): 23
-----
Сума бројева до 1000000000 је 499999999500000000
Време у ns са nanoTime(): 228048000
Време у ms са nanoTime(): 228.05
Сума бројева до 1000000000 је 499999999500000000
Време у ms са currentTimeMillis(): 228
-----
```

6.1.3. Захтев за покретањем скупљача отпадака, метод `System.gc()`

Као што је раније речено, Јава аутоматски уклања са хипа податке који више нису у употреби, тј. на које се више не референцира. Скупљач отпадака се активира по потреби и његов рад се заснива на коришћењу разних хеуристика. Захваљујући хеуристикама, скупљач отпадака се прилагођава конкретном процесу, предвиђа употребу меморије и слично. Квалитет скупљача отпадака може се оценити коришћењем два критеријума:

1. колико је, у просеку, неки објекат „чекао“ у меморији од момента када је постао непотребан до момента уклањања и
2. колики ефекат на перформансе целе виртуелне машине има рад скупљача отпадака.

Ова два критеријума су често у колизији, тј. минимизација времена чекања на уклањање подразумева чешће активације скупљача отпадака и самим тим смањење перформанси виртуелне машине.

(Напомена: добро написан С програм, у којем се меморија експлицитно ослобађа чим више није неопходна (наредбом `free()`), нуди максималне перформансе уз минимизацију ефекта „чекања“.)

Програмер, дакле, у Јави не може уклањати податке из меморије самостално. Највише што може је да сугерише скупљачу отпадака да се активира преко метода `System.gc()`. Ретке су ситуације у којима програмер може да има користи од експлицитног позивања овог метода и генерално се његово позивање не препоручује.

Написати Јава програм који користи једну референцу променљиву за прављење великог броја нових ниски на хипу. Ово за последицу има велики број нерелевантних објеката током рада програма. Упоредити количину доступне и слободне меморије на хипу у варијантама:

1. када скупљач сам одређује кад ће се активирати и
2. када му корисник повремено сугерише скупљање помоћу `System.gc()` метода.

```
public class PozoviGC
{
    static void stanjeMemorije() {
        long slobodno = Runtime.getRuntime().freeMemory()/1024/1024;
        long maksimalno = Runtime.getRuntime().totalMemory()/1024/1024;
        System.out.printf("%dMB од %dMB | ", slobodno, maksimalno);
    }

    public static void main(String[] args) {
        int n = 10000000;
        int nIspis = 200000;
        String s;
        long pocBezGC = System.nanoTime();
        System.out.println(
            "Величине слободне меморије без позивања gc() пред испис");
        for (int i = 0; i < n; i++) {
            s = new String("Текст под редним бројем " + i);
            if (i % nIspis == 0)
                stanjeMemorije();
        }
    }
}
```

```

    }
    System.out.printf("%nВреме без GC\t%.2f%n",
        (System.nanoTime() - pocBezGC) / 1e6);

    long pocSaGC = System.nanoTime();
    System.out.println(
        "Величине слободне меморије са позивањем gc() пред испис");
    for (int i = 0; i < n; i++) {
        s = new String("Текст под редним бројем " + i);
        if (i % nIspis == 0) {
            System.gc();
            stanjeMemorije();
        }
    }
    System.out.printf("%nВреме са GC\t%.2f%n",
        (System.nanoTime() - pocSaGC) / 1e6);
}
}

```

У варијанти без употребе `System.gc()` метода види се да програм ради са већом (флексибилнијом) количином меморије, што омогућава да скупљач не мора често да се активира. Тек повремено, приликом значајног смањења удела слободне меморије, скупљач се самостално активира и обрише непотребне податке. Са друге стране, у режиму експлицитног позивања скупљача отпадака, количина слободне меморије је врло близу максимуму. Време извршавања је у другом случају нарушено, што је очекивана последица скупог процеса скупљања отпадака.

Следи резултат рада програма.

```

Величине слободне меморије без позивања gc() пред испис
1021МВ од 1024МВ | 978МВ од 1024МВ | 987МВ од 1024МВ | 973МВ од 1024МВ |
957МВ од 1024МВ | 941МВ од 1024МВ | 927МВ од 1024МВ | 1015МВ од 1024МВ |
999МВ од 1024МВ | 983МВ од 1024МВ | 967МВ од 1024МВ | 955МВ од 1024МВ |
939МВ од 1024МВ | 923МВ од 1024МВ | 907МВ од 1024МВ | 891МВ од 1024МВ |
879МВ од 1024МВ | 863МВ од 1024МВ | 847МВ од 1024МВ | 831МВ од 1024МВ |
815МВ од 1024МВ | 799МВ од 1024МВ | 787МВ од 1024МВ | 771МВ од 1024МВ |
755МВ од 1024МВ | 739МВ од 1024МВ | 723МВ од 1024МВ | 711МВ од 1024МВ |
695МВ од 1024МВ | 679МВ од 1024МВ | 663МВ од 1024МВ | 647МВ од 1024МВ |
631МВ од 1024МВ | 619МВ од 1024МВ | 603МВ од 1024МВ | 587МВ од 1024МВ |
571МВ од 1024МВ | 555МВ од 1024МВ | 543МВ од 1024МВ | 527МВ од 1024МВ |
511МВ од 1024МВ | 495МВ од 1024МВ | 479МВ од 1024МВ | 463МВ од 1024МВ |
451МВ од 1024МВ | 435МВ од 1024МВ | 1019МВ од 1024МВ | 1003МВ од 1024МВ |

```



```
987MB од 1024MB | 975MB од 1024MB |  
Време без GC      373.09  
Величине слободне меморије са позивањем gc() пред испис  
55MB од 56MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB  
| 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од  
32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB  
од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB |  
31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB  
| 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од  
32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB  
од 32MB | 31MB од 32MB | 55MB од 56MB | 31MB од 32MB | 31MB од 32MB |  
31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB  
| 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB | 31MB од  
32MB | 31MB од 32MB | 31MB од 32MB | 31MB од 32MB |  
Време са GC      445.92
```

6.1.4. Излазак из апликације, метод `System.exit()`

Попут функције `exit()` у програмском језику С и Јава има могућност да прекине извршавање актуелног програма. У Јави се, заправо, гаси читава Јава виртуелна машина. Метод је статички и налази се у класи `System`. Његово заглавље је:

```
public static void exit(int status)
```

Статус 0 указује на успешан завршетак извршавања док не-нула вредности указују на неуспешно извршавање, тј. на грешку.

Написао Јава програм који реализује и тестира метод за исписивање карактеристика монитора, при чему су аргументи метода ширина и висина. Потребно је израчунати број тачака (пиксела) и сврстати монитор у категорију стандардни или широки на основу односа ширине и висине: монитор је широк уколико је ширина два или више пута већа од висине. Такође је потребно на одговарајући начин реаговати на некоректне уносе.