

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/303858129>

Projektovanje softvera – Napredne Java tehnologije (Software design – Advanced Java Technologies)

Book · January 2008

CITATIONS

0

READS

3,762

5 authors, including:



Siniša Vlajić

Faculty of organisational sciences - Universit...

43 PUBLICATIONS 28 CITATIONS

[SEE PROFILE](#)



Dusan Savic

University of Belgrade

26 PUBLICATIONS 20 CITATIONS

[SEE PROFILE](#)



Stanojevic Vojislav

University of Belgrade

18 PUBLICATIONS 17 CITATIONS

[SEE PROFILE](#)



Ilija Antović

University of Belgrade

20 PUBLICATIONS 18 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



SilabMDD [View project](#)

УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

Одсек за информационе системе и технологије – Катедра за софтверско инжењерство

Лабораторија за софтверско инжењерство

ПРОЈЕКТОВАЊЕ СОФТВЕРА – НАПРЕДНЕ ЈАВА ТЕХНОЛОГИЈЕ

Аутори:

СИНИША ВЛАЈИЋ, ДУШАН САВИЋ
ВОЈИСЛАВ СТАНОЈЕВИЋ
ИЛИЈА АНТОВИЋ, МИЛОШ МИЛИЋ



Београд – 2008.

УНИВЕРЗИТЕТ У БЕОГРАДУ
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

Одеј за информационе системе и технологије – Катедра за софтверско инжењерство

Лабораторија за софтверско инжењерство

**ПРОЈЕКТОВАЊЕ СОФТВЕРА –
НАПРЕДНЕ ЈАВА ТЕХНОЛОГИЈЕ**

Аутори:

**СИНИША ВЛАЈИЋ, ДУШАН САВИЋ
ВОЈИСЛАВ СТАНОЈЕВИЋ
ИЛИЈА АНТОВИЋ, МИЛОШ МИЛИЋ**



ЗЛАТНИ ПРЕСЕК

Београд – 2008.

Аутори

**Синиша Влајић, Душан Савић, Војислав Станојевић, Илија Антовић и
Милош Милић**

Прво издање

Наслов

ПРОЈЕКТОВАЊЕ СОФТВЕРА – НАПРЕДНЕ ЈАВА ТЕХНОЛОГИЈЕ

Издавач

Агенција за пружање интелектуалних услуга издаваштво и трговину “Златни пресек”

Адреса издавача

Илије Ђуричића 56, Београд

Тел/факс: (011-502-872)

Е-пошта: info@zlatnipresek.co.yu

Web адреса: www.zlatnipresek.co.yu

Главни уредник

Сузана Влајић

Рецензент

Др Владан Девецић, ред. проф.

Припрема и дизајн

Аутори

Коректура

Аутори

Тираж

400

Штампано

2008

Штампа

Штампарија “Јуниор”, Београд

[СИР – Каталогизација у публикацији
Народна библиотека Србије, Београд

004.41(075.8)

004.438JAVA(075.8)

ПРОЈЕКТОВАЊЕ софтвера: напредне Јава
технологије / аутори Синиша Влајић ... [и др.].
– 1. изд. – Београд : Златни пресек,
2008 (Београд : Јуниор). – 215 стр. : илустр.
; 30 cm

На врху насл. стр. : Универзитет у Београду,
Факултет организационих наука, Одсек за
информационе системе и технологије, катедра
за софтверско инжењерство, Лабораторија за
софтверско инжењерство. – Тираж 400. –
Библиографија: стр. 215.

ISBN 978-86-86887-03-0

1. Влајић, Синиша [автор]

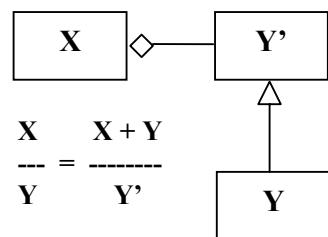
а) Софтвер – Пројектовање б) Програмски језик
“Јава”

COBISS.SR-ID 146593036]

Copyright © Агенција Златни пресек

БИБЛИОТЕКА
СОФТВЕРСКО ИНЖЕЊЕРСТВО

УРЕДНИК
СИНИША ВЛАЈИЋ



ЗЛАТНИ ПРЕСЕК

УВОД

У јесен 2005 године мала али одабрана екипа са ФОН-а је започела да ради неформални интерни пројекат под називом: *Софтверско инжењерство као научна дисциплина*. Циљ тог пројекта се односио на сагледавање и схватање софтверског инжењерства као легитимне инжењерске дисциплине и професије. У том смислу је посебна пажња усмерена на сагледавање основних области знања софтверског инжењерства као што су: *прикупљање захтева, пројектовање софтвера, конструкција софтвера, ..., квалитет софтвера*.

Једна од најважнијих области знања софтверског инжењерства је *пројектовање софтвера*. У том смислу, код објашњења процеса животног циклуса софтвера по ISO-IAC 12207 стандарду, пројектовање софтвера се састоји од две активности:

- *Пројектовање софтверске архитектуре* (понекад се назива пројектовање највишег нивоа (top-level design)), описује структуру и организацију софтвера на највишем нивоу. На овом нивоу се идентификују различите компоненте.
- *Детаљно пројектовање софтвера* које описује сваку компоненту до нивоа детаљности који је довољан да се она може конструисати.

Ова књига треба да покрије активности детаљног пројектовање софтвера коришћењем напредних Јава технологија J2SE платформе.

Ова књига се састоји од 10 тематских јединица. У даљем тексту ћемо укратко објаснити сваку од наведених тематских јединица и имена аутора који су за њих били задужени:

У **првој глави** књиге (С. Влајић, Д. Савић) биће објашњени основни концепти Јаве (*класе и методе, наслеђивање, апстрактне класе, интерфејси, изузеци, пакети и UI токови*). Наведени концепти су предуслов за схватање напредних Јава технологија.

У **другој глави** књиге (С. Влајић) биће објашњено конкурентно програмирање у Јави коришћењем нити. У том смислу биће објашњено како се праве, прекидају, комуницирају и синхронизују нити.

У **трећој глави** (С. Влајић) ће бити размотрено програмирање у мрежи како би се направиле једноставне клијент/сервер апликације. Посебно ће бити наглашен концепт сокета као кључног механизма у мрежном програмирању.

У четвртој глави (С. Влајић) ће бити представљена RMI (Remote Method Invocation) технологија која омогућава позив метода удаљених објеката у мрежи.

У петој глави (С. Влајић) ће бити размотрени неки од механизама заштите у Јави.

У шестој глави (С. Влајић) ће бити објашњено како се Јава програми повезују са Системима за управљање базама података и биће укратко објашњен један скуп SQL упита који су направљени у оквиру MySQL система за управљање базом података.

У седмој глави (Д. Савић) ће бити биће објашњени неки од основних графичких елемената AWT и SWING графичке библиотеке у Јави.

У осмој глави (В. Станојевић) ће бити објашњена рефлексија као механизам који омогућава добијање основних (мета)информација о класама и интерфејсима.

У деветој глави (М. Милић) ће бити представљене JAXP и JAXB Јавине технологије које подржавају рад са XML документима.

У десетој глави (И. Антовић) ће бити размотрена JAX-WS2.0 Јавина библиотека која подржава рад са Web сервисима.

Ова књига је у ужем смислу намењена за предмет *Пројектовање софтвера* који се изучава на основним и мастер студијама на ФОН-у.

У ширем смислу ова књига је намењена свима онима који се баве развојем и пројектовањем системског и апликативног софтвера коришћењем Јава технологија.

Овом приликом се захваљујемо рецензенту ове књиге професору Др Владану Девеџићу који је пружио значајну подршку у писању ове књиге.

АУТОРИ

1. ЈАВА ТЕХНОЛОГИЈА.....	1
1.1 Јава платформа	1
1.1.1 Java 2 Platform, Standard Edition или J2SE	1
1.1.2 Java 2 Platform, Enterprise Edition или J2EE.....	2
1.1.3 Java 2 Platform, Micro Edition или J2ME	3
1.2 Објектно-оријентисани програмски језик Јава	4
1.2.1 Класе и методе.....	5
1.2.2 Наслеђивање	10
1.2.3 Апстрактне класе	13
1.2.4 Интерфејси.....	14
1.2.5 Изузети.....	16
1.2.6 Пакети.....	19
1.2.7 Улазно-излазни токови.....	21
2. НИТИ.....	27
2.1 Главна програмска нит	27
2.2 Прављење нити	28
2.2.1 Прављење нити реализацијом класе Runnable	28
2.2.2 Прављење нити проширењем класе Thread.....	32
2.3 Прављење више нити	32
2.4 Стана нити.....	33
2.5 Прекид нити.....	35
2.6 Коришћење метода isAlive() и join().....	41
2.7 Приоритет извршавања нити.....	44
2.8 Себичне (<i>selfish</i>) нити.....	45
2.9 Групе нити	45
2.10 Синхронизација	48
2.10.1 Комуникација нити без синхронизације	48
2.10.2 Закључавање објекта	49
2.10.3 Коришћење синхронизованих метода	50
2.10.4 Коришћење синхронизованих објекта.....	51
2.11 Комуницирање између нити	52
2.12 Међусобно блокирање нити	60
2.13 Коришћење цеви (pipes) за комуникацију између нити.....	62
3. ПРОГРАМИРАЊЕ (РАД) У МРЕЖИ.....	63
3.1 Адреса рачунара.....	63
3.2 URL адреса	64
3.3 Сокети	66
3.3.1 Адреса сокета	66
3.3.2 Конекција	66
3.3.3 Повезивање сервера са више клијената	70
3.4 Слање електронске поште	81
4. РАД СА УДАЉЕНИМ ОБЈЕКТИМА.....	84
4.1 Улоге клијента и сервера	84
4.2 Позив удаљених метода	84
4.2.1 Комуникација клијентског и серверског објекта	84
4.2.2 Веза између клијентског и серверског објекта.....	86
4.2.3 Структура серверског програма.....	86
4.2.3.1 Регистровање серверских објеката	87
4.2.4 Структура клијентског програма	88
4.2.4.1 Повезивање клијента са стуб класом.....	88
4.2.5 Повезивање клијентског са серверским објектом.....	89
4.2.6 Примери за RMI	90
4.2.7 Генерички RMI пример	95
5. ЗАШТИТА.....	98
5.1 Пуњачи класа.....	98
5.1.1 Писање сопственог пуњача класа	99
5.1.2 Енкриптовање класе	100
5.1.3 Пример пуњача класе који памти енкриптовану класу.....	101
5.2 Byte code верификација.....	103
5.2.1 Промена class датотека преко хекс едитора	103
5.2.2 Рекомпајлери class датотека	106
5.3 Менаџери заштите (Security managers) и дозволе (permissions).....	106

5.4 Дигитални потпис	108
5.5 Аутентификација	109
5.6 Означавање аплета.....	109
6. РАД СА БАЗОМ – JDBC	115
6.1 Поступак повезивања Јава програма и СУБП-а.....	115
6.2 Поступак извршења операција над базом података СУБП	118
6.3 Примери SQL упита над MySQL СУБП	125
6.3.1 Дефиниција података (data definition)	125
6.3.2 Манипулација подацима (data manipulation)	126
7. ГРАФИЧКИ КОРИСНИЧКИ ИНТЕРФЕЈС У ЈАВИ.....	136
7.1 Креирање графичких корисничких форми	138
7.2 Управљање догађајима.....	141
7.2.1 Концепт догађаја	141
7.2.2 Класе догађаја.....	141
7.2.1.1 Класе семантичких догађаја	141
7.2.1.2 Класе догађаја ниског нивоа.....	142
7.2.2 Слушаоци догађаја.....	142
7.2.2.1 Слушаоци семантичких догађаја	143
7.2.2.2 Слушаоци догађаја ниског нивоа.....	147
7.3 Адаптер класе	153
7.4 Рад са компонентом JPanel	155
7.5 Рад са дефинисаним дијалозима	158
7.6 Рад са менијима.....	161
8. МЕХАНИЗАМ РЕФЛЕКСИЈЕ У ЈАВИ	166
8.1 Добијање основних информација о класи коришћењем Class објекта	166
8.1.2 Испитивање атрибута класе коришћењем рефлексије	166
8.1.3. Испитивање метода у времену извршавања	167
8.2. Коришћење Field класе за рад са атрибутима објеката	168
8.2.1 Постављање и узимање вредности атрибута.....	169
8.2.2 Испитивање и рад са модификаторима	170
8.2.3 Приступ не-јавним чланцима класе	170
8.3. Коришћење Method klase за рад са методама	172
8.3.1 Динамичко позивање метода класа	172
8.3.1.1 Коришћење простих типова приликом динамичог позива метода класа	173
8.4. Динамичко учитавање и креирање објеката коришћењем рефлексије	174
8.4.1 Основе forName методе	174
8.4.2 Креирања инстанце класе коришћењем newInstance методе, класе Class	174
8.4.3 Креирања инстанце класе коришћењем Constructor класе	175
8.5 Пролазак кроз хијархију наслеђивања	176
8.5.1 Испитивање хијерархије наслеђивања	177
8.6. Рад са мета-подацима табела у бази	178
9. ЈАВА И XML.....	181
9.1 Обрада XML докумената	182
9.1.1 SAX	182
9.1.2 DOM	185
9.1.3 JAXB	190
9.2 XSLT трансформација	195
9.3 Питања.....	198
9.4 Задаци за вежбу	199
10. WEB СЕРВИСИ	201
10.1 Web сервиси у Java SE 6–JAX WC	203
10.1.1 Кораци при креирању web сервиса	204
10.1.2 Кораци при креирању клијента web сервиса	208
10.1.3 Објављивање web сервиса на Apache Tomcat web серверу	210
10.2 Питања	214
10.3 Задаци за вежбу	214
Референце	215

ПРОЈЕКТОВАЊЕ СОФТВЕРА – НАПРЕДНЕ ЈАВА ТЕХНОЛОГИЈЕ

И који хоће први међу вама да буде, да буде свима слуга. Јер Син човечји није дошао да My служе него да служи, и да да душу своју у откуп за многе.

(Јеванђеље по Марку 10:44-45)

Пројектовање софтвера

1. ЈАВА ТЕХНОЛОГИЈА

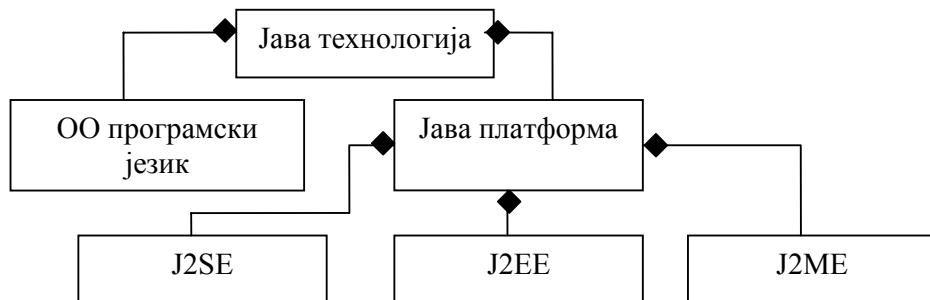
Јава је софтверска технологија која је у исто време и платформа и објектно-оријентисани програмски језик [R2D1J2EEKOMP] (Слика JT).

1.1 Јава платформа

Јава Платформа је назив за рачунарско окружење (*computing environment*) или платформу *Sun Microsystems-a* у коме се може извршити апликација која је развијена коришћењем Јава програмског језика и скупа развојних оруђа (*development tools*). У овом случају платформа није специфичан хардвер или оперативни систем. То је извршни механизам (*execution engine*) који је реализован преко витуелне машине (*virtual machine*) и скупа стандардних библиотека које обезбеђују жељену функционалност [WikiJavaPlatform].

Јава платформа укључује (Слика JT):

- a) *Java 2 Platform, Standard Edition* или *J2SE*
- b) *Java 2 Platform, Enterprise Edition* или *J2EE*
- c) *Java 2 Platform, Micro Edition* или *J2ME*



Слика JT: Јава технологија

1.1.1 Java 2 Platform, Standard Edition или J2SE

Java 2 Platform, Standard Edition или *J2SE* је развојно окружење које обезбеђује Јава API (*Application Programming Interface*) и оруђа за креирање, тестирање и извршавање Јава програма (апликација, аплета и компоненти). *J2SE* се састоји из два основна производа [<http://java.sun.com/j2se/overview.html>] (Слика J2SE):

a) Java SE Runtime Environment (JRE)

JRE се састоји из Јава Вируелне Машине (*Java Virtual Machine - JVM*), J2SE API-а и скупа технологија распоређивања (*deployment technologies*) као што су нпр. *Java Web Start* и *Java Plug-in* које подржавају извршење Јава програма. J2SE API се састоји од много технологија¹: AWT, Swing, JDBC, RMI, JAXP, JAXB...,JNDI. JRE је основа за J2EE технологију.

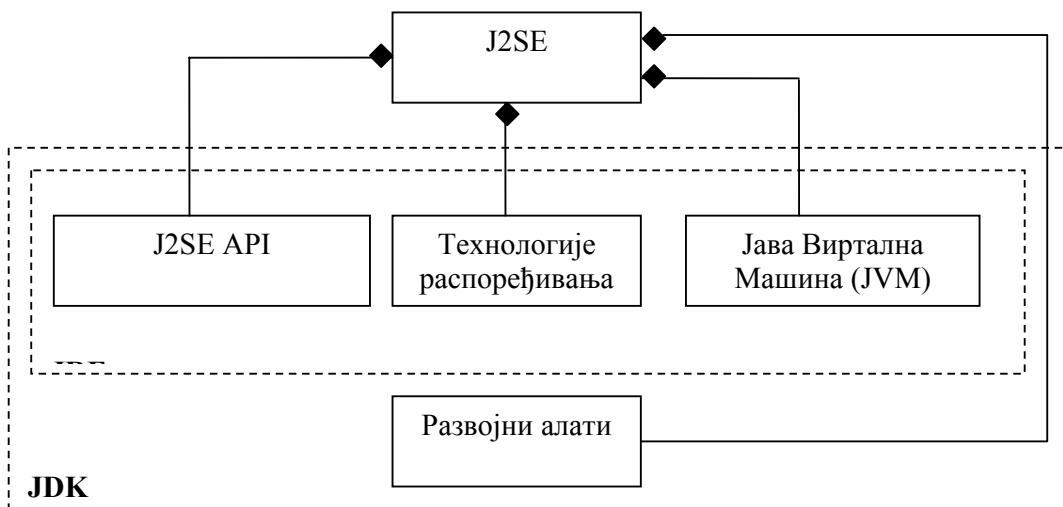
¹ Јава технологије су дефинисане одговарајућим спецификацијама. *Java Community Process* (JCP) је одговоран за развој спецификација Јава технологија <http://jcp.org>.

b) Java SE Development Kit (JDK)

JDK обезбеђује развојне алате (компајлере (*compilers*) и дибагере (*debuggers*)) за развој и тестирање Јава програма. JRE је укључен у JDK.

J2SE verzije: J2SE 1.2,..., J2SE 1.4 (*Merlin*), J2SE 5.0 (*Tiger*), Java SE 6 (*Mustang*),...
<http://java.sun.com/products/archive/index.html>, <http://jcp.org/en/jsr/tech?listBy=2&listByType=platform>.

Нови назив за наведену платформу је **Java Platform, Standard Edition** или **Java SE**.



Слика J2SE: Java 2 Platform, Standard Edition (J2SE)

1.1.2 Java 2 Platform, Enterprise Edition или J2EE

Java 2 Platform, Enterprise Edition или **J2EE** је развојно окружење које обезбеђује J2EE API и оруђа за грађење, тестирање и извршавање J2EE апликација². J2EE је заснована на J2SE³. J2EE API се састоји од много технологија: JSP, Java Servlet, JSF, EJB,..., JAXP. J2EE технологије су подељене у три групе [J2EE14TUT](Слика J2EE):

a) Јава Web технологија

Web технологије су:

- *Java Servlet*
- *JavaServer Pages (JSP)*
- *JavaServer Pages Standard Tag Library (JSTL)*
- *Java Server Faces (JSF)*
- Интернационализација и локализација Web апликација (*Web application internationalization and localization*)

b) Enterprise JavaBeans (EJB) технологија

EJB технологије су:

- *Session beans*

² J2EE апликације су вишенивојске сложене апликације (*multitier enterprise applications*).

³ J2EE користи многе карактеристике J2SE као што су JDBC API за рад са базом података, JNDI технологију за интеракцију између enterprise ресурса и модела заштите података (*data security model*)

- *Entity beans*
- *Message-driven beans*

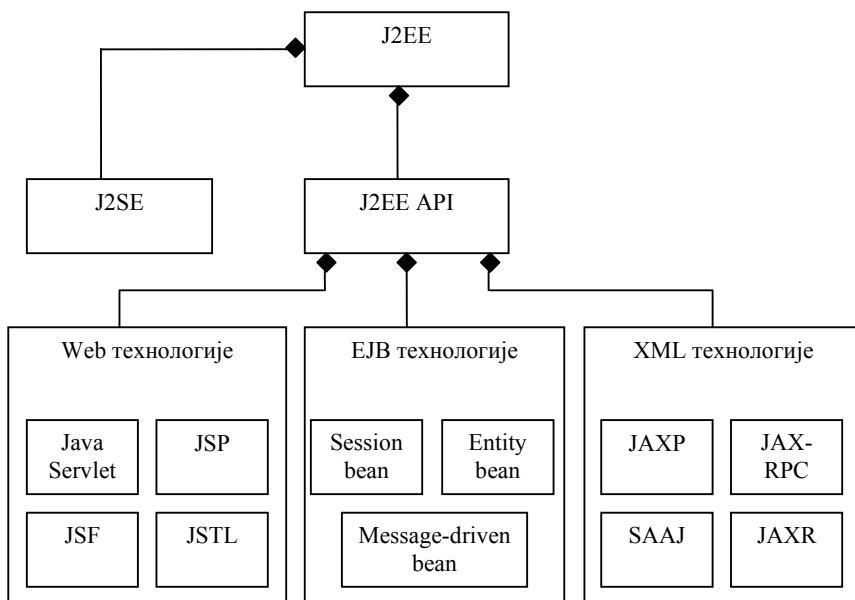
c) Java XML технологија

Java XML технологије су:

- *The Java API for XML Processing (JAXP)*
- *The Java API for XML based RPC (JAX-RPC)*
- *SOAP with Attachments API for Java (SAAJ)*
- *The Java API for XML Registries (JAXR)*

J2EE платформа подржава:

- Више нивојски дистрибуирани апликациони модел (*Multitiered distributed application model*).
- Компоненте које се могу поново користити (*Reusable components*).
- Јединствени модел заштите (*Unified security model*).
- Флексибилну трансакциону контролу (*Flexible transaction control*).
- Web сервисе који прихватају и размењују податке који су засновани на XML (*Extensible Markup Language*) стандардима и протоколима.



Слика J2EE: Java 2 Platform, Enterprise Edition (J2EE)

J2EE верзије: ...,J2EE 1.3, J2EE 1.4, Java EE 5.0,...

Свака од J2EE верзија је подржана одговарајућим технологијама. На пример, J2EE 1.4 је подржана са JSP 2.0, Java Servlet 2.4, ..., EJB 2.1 технологијама. Java EE 5.0 је подржана са JSP 2.1, Java Servlet 2.5, ..., EJB 3.0 технологијама;

Нови назив за наведену платформу је **Java Platform, Enterprise Edition** или **Java EE**.

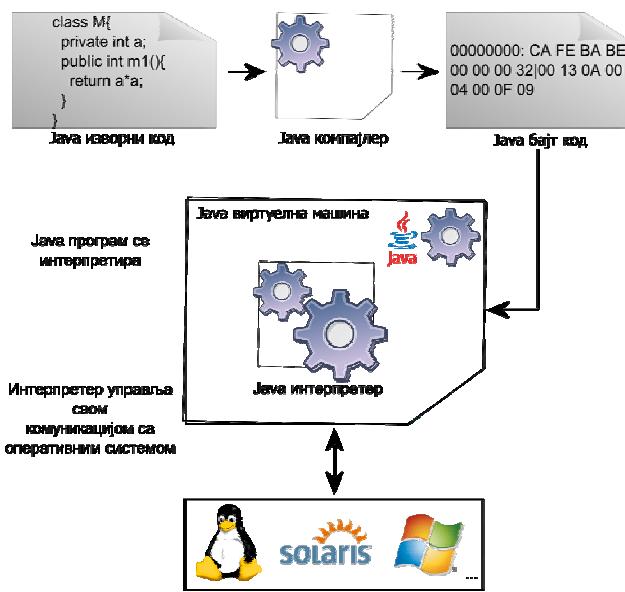
1.1.3 Java 2 Platform, Micro Edition или J2ME

Java 2 Platform, Micro Edition или **J2ME** је развојно окружење које обезбеђује API (Bluetooth, TV, USB,...) и оруђа за грађење, тестирање и извршавање J2ME апликација које се користе за уређаје као што су PDA или мобилни телефони.

Нови назив за наведену платформу је **Java Platform, Micro Edition** или **Java ME**.

1.2 Објектно-оријентисани програмски језик Јава

Јава је објектно оријентисани програмски језик јер подржава концепте објектно-оријентисаног програмирања као што су: класе, објекте, наслеђивање, учаурење, јак полиморфизам, преклапање и прекривање метода, интерфејсе, апстрактне класе,...,итд. Исти Јава програм се може извршити на различитим оперативним системима (*Windows, Solaris, Linux...*). Предуслов извршења Јава програма на неком оперативном систему је постојање Јава виртуелне машине за тај оперативни систем. То значи да се за сваки оперативни систем прави посебна Јава виртуална машина. Улога Јава виртуелне машине јесте да преслика наредбе Јава програма (*bytecode* наредбе) у наредбе конкретног оперативног система. Поступак компајлирања и извршавања Јава програма се може представити преко слике *ПЈ-Јава1*.



Слика ПЈ-Јава1: Начин извршавања Јава програма

Јава програм⁴ се компајлира помоћу *Java компајлера* и као резултат се добијају Јава бајт код (*bytecode*) наредбе. Јава бајт код наредбе се затим интерпретирају помоћу Јава виртуелне машине⁵. Као што је већ речено Јава виртуелна машина пресликава Јава бајт код наредбе у наредбе оперативног система

За писања Јава програма је потребно да постоји инсталација неки од текст едитора⁶ или неко од развојних окружења (*NetBeans, Eclipse ...*).

За компајлирање Јава програма на рачунару је потребно да постоји инсталација Јава *SDK* (*Standard Development Kit*). За извршење Јава програма је потребно да постоји инсталација Јава *JRE*(*Java Runtime Environment*)⁷.

У наставку ће укратко бити објашњено компајлирање и извршење извornог кода конкретне Јава датотеке *M.java*:

⁴ Јава програм садржи Јава наредбе које су у извornом облику (извornи програмски код) који се налази у Јава датотеци (датотека са екstenзијом *.java*).

⁵ Приликом преузимања (*download*) неких од интернет претраживача (*web browser*) могуће је преузети и Јава виртуелну машину заједно са њима. Јава виртуелна машина је потребна да постоји на рачунарима како би се извршавали аплети унутар интернет претраживача.

⁶ Препорука је да се користе текст едитори који препознају синтаксу програмског језика Јава, као што је на пример *Textpad*.

⁷ Већ је наглашено да је JRE део Јава SDK. То значи да је за компајлирање и извршење Јава програма довољно да буде инсталација Јава SDK.

1. Компајлирање Јава програма – Помоћу наредбе *javac* се врши превођење Јава извornог кода у Јава бајт код:

```
javac M.java
```

Као резултат компајлирања се добија датотека/е са екstenзијом *.class*. Број *.class* датотека зависи од броја класа које се налазе у датотеци *M.java*.

Уколико се у датотеци *M.java* налата само једна класа (класа M), тада ће се као резултат компајлирања добити датотека *M.class*.

2. Извршење Јава програма – Помоћу наредбе *java* се извршава Јава програм који је у бајт код облику:

```
java M
```

Као резултат ове наредбе се извршава *main()* метода класе M.

1.2.1 Класе и методе

Класа представља апстрактну представу скупа објеката који имају исте особине. Класа се састоји од **атрибута**⁸ и **метода**. Атрибутима се дефинише понашање, а методама понашање класе. Заједно (атрибути и методе класе) се називају чланице класе.

Објекат представља једно конкретно појављивање (примерак, инстанцу) своје класе. Низе је дат општи облик дефинисања класе:

```
class <имекласе>{
    <тип> <атрибут1>;
    ...
    <тип> <атрибутn>;
    <тип> <име_метод1>(<листа параметара>{
        тело методе 1
    })
    ...
    <тип> <име_методем>(<листа параметара>{
        тело методе м
    })
}
```

Методе класе

Методама се дефинише понашање класе.

Свака метода садржи:

- тип који враћа (или *void* уколико не враћа никакав тип),
- назив,
- листу параметара и
- тело методе.

Уколико метода враћа неку вредност тада се користи кључна реч *return* преко које метода враћа вредност.

```
public int saberi(){
    return a+b;
}
```

У овом примеру метода враћа целибројну вредност (*int*) која представља збир вредности које садрже атрибути a и b.

Уколико метода садржи параметре, за њу се каже да је она параметризована метода.

```
public void postaviVrednosti(int a1, int b1){
    a = a1;
    b = b1;
}
```

⁸ У Јави се користи термин *instance variables* када се жели указати на атрибуте класа.

Пример ОК-1 - Дефинисати класу CSabiranje. Класа саджи два атрибута (*a* и *b*) целобројног типа (*int*) и две методе од којих једна метода (*postaviVrednosti*) поставља нову вредност атрибутима класе и друга (*saberij*) која рачуна збир вредности ова два атрибута.

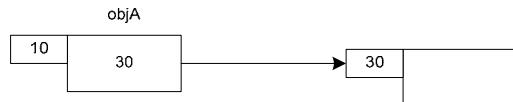
```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class CSabiranje{
    /* atributi klase */
    int a;
    int b;
    /* metode klase */
    public void postaviVrednosti(int a1, int b1){
        a = a1;
        b = b1;
    }
    public int saberi(){ return a+b; }
}
```

Декларација објекта

Објекат представља једно појављивање класе. Објекат се декларише и креира на следећи начин:

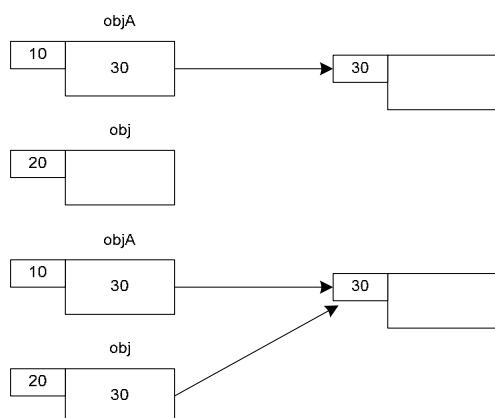
```
CSabiranje objA = new CSabiranje();
```

objA је у суштини референца на објекат а не сам објекат.



Референца на објекат `objA` може да добије референцу на било који објекат који је класе `CSabiranje` (или класа које су изведене⁹ из класе `CSabiranje`). На пример:

```
CSabiranje objA = new CSabiranje();
CSabiranje obj;
obj = objA;
```



За објекте `objA` и `obj` каже се да су референтни објекти јер они садрже референцу, односно адресу објекта.

Пример ОК-2. Дефинисати класу `Test`. Класа `Test` садржи `main()` методу. У `main()` методи креирати објекат (`objA`) класе `CSabiranje` из претходног примера. Креирати референтни објекат `obj` и доделити му референцу креираног објекта (`objA`).

⁹ Погледати поглавље наслеђивање које објашњава овај концепт

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class Test{
    public static void main(String[] args){
        CSabiranje objA = new CSabiranje();
        CSabiranje obj;
        obj = objA;
    }
}
```

Конструктори

Конструктори представљају специјалне методе помоћу којих се врши иницијализација објекта (додељивање почетних вредности атрибутима објекта), приликом њиховог креирања.

Наводимо неке од основних особина конструктора:

- Конструктор има исти назив као и име класе.
- Конструктор се покреће код декларације објекта.
- Конструктори не враћају никакав тип података (чак ни `void`).

Подразумевани (default) конструктори

Подразумевани (*default*) конструктори постављају вредности атрибути сваког објекта на исту почетну вредност.

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class Test{
    public static void main(String[] args){
        CSabiranje objA = new CSabiranje();
        CSabiranje objB = new CSabiranje();
    }
}
```

```
class CSabiranje{
    /* atributi klase */
    int a;
    int b;
    /* podrazumevani (default) konstruktor */
    CSabiranje (){
        a = 0;
        b = 9;
    }
    /* metode klase */
    public void postaviVrednosti(int a1, int b1){
        a = a1;
        b = b1;
    }
    public int saberi(){
        return a+b;
    }
}
```

У овом примеру приликом креирања објекта `objA` и `objB` позива се подразумевани конструктор који додељује исту вредност ($a=0$, $b=9$) атрибутима објеката `objA` и `objB`.



Параметарски конструктор

Параметарски конструктори постављају атрибуте објекта на вредности које су дефинисане параметрима код декларације објекта.

```

/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class Test{
    public static void main(String[] args){
        /* poziva se default konstruktor */
        CSabiranje objA = new CSabiranje();

        /* poziva se parametrizovani konstruktor */
        CSabiranje objB = new CSabiranje(10,20);
    }
}
    
```

```

class CSabiranje{
    /* atributi klase */
    int a;
    int b;
    /* podrazumevani konstruktor */
    CSabiranje (){
        a = 0;
        b = 9;
    }
    /* parametrizovani konstruktor */
    CSabiranje (int a1, int b1){
        a = a1;
        b = b1;
    }
    public int saberi(){
        return a+b;
    }
}
    
```

У овом примеру приликом креирања објекта objA позива се подразумевани конструктор који атрибутима *a* и *b* додељује вредности 0 и 9. Такође, приликом креирања објекта objB позива се параметризовани конструктор који атрибутима *a* и *b* додељује вредности 10 и 20.



Преклапање метода

У Јави као и у C++ могуће је дефинисати истоимене методе у оквиру једне класе. У том случају се каже да су методе преклопљене (*overloaded*). Преклапање метода представља један облик полиморфизма (полиморфно - нешто што има више облика). Међутим у оквиру једне класе преклопљене методе морају да се разликују по потпису (по броју и/или типу параметара).

Пример дефинисања преклопљених метода:

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class CSabiranje{
    /* atributi klase */
    int a;
    int b;
    /* default konstruktor */
    CSabiranje (){
        a = 0;
        b = 9;
    }
    /* parametrizovani konstruktor */
    CSabiranje (int a1, int b1){
        a = a1;
        b = b1;
    }
    public int saberi(){
        return a+b;
    }
    public int saberi(int a, int b){
        return a+b;
    }
}
```

У овом примеру дефинисане су две методе `saberi`. За ове две методе каже се да су преклопљене јер имају исто име, а различит потпис.

Преклапање конструктора

Као и било које друге методе тако се и конструктор методе могу преклопити.

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class CSabiranje{
    /* atributi klase */
    int a;
    int b;
    /* default konstruktor */
    CSabiranje (){
        a = 0;
        b = 9;
    }
    /* parametrizovani konstruktor */
    CSabiranje (int a1, int b1){
        a = a1;
        b = b1;
    }
    public int saberi(){
        return a+b;
    }
    public int saberi(int a, int b){
        return a+b;
    }
}
```

У овом примеру класа `CSabiranje` има два преклопљена конструктора и то: *default* конструктор и један параметарски конструктор.

Задаци за вежбање:

Задатак ОК1: Дефинисати класу `Drzava` која садржи атрибуте `naziv`, `glavniGrad`, `povrsina`, `brojStanovnika`. Креирати објекта класе `Drzava` и поставити вредности атрибута преко параметризованог конструктора. Дефинисати и позвати методу за приказ вредности наведених атрибута.

Задатак ОК2: Ниже је дат програмски код који је потребно допунити. Дефинисати класу `Krug` и одговарајуће методе.

```
class Glavna{
    public static void main(String[] args){
        Krug k = new Krug();
        k.postaviPoluprecnik(12.7);
        double p = k.izracunajPovrsinu();
        System.out.println("Povrsina je: "+p);
    }
}
```

Питања :

1. Објаснити разлику између објекта и класе.
2. Објаснити концепт преклапања метода.
3. Објаснити улогу параметризованих конструктора.
4. Да ли су наведене методе унутар класе `Primer1` исправно дефинисане.Објаснити зашто.

```
class Primer1{
    void m1(){
    }
    int m1(){
        return 0;
    }
}
```

5. Да ли су наведене методе унутар класе `Primer2` исправно дефинисане.Објаснити зашто.

```
class Primer2{
    void m1(int a, int b){
    }
    int m1(int a){
        return a+10;
    }
}
```

1.2.2 Наслеђивање

Наслеђивање представља један од основних концепата објектно-оријентисаног програмирања. Помоћу концепта наслеђивања, класа која је изведена из основне класе, наслеђује све атрибуте и методе основне класе. Основну класу називамо надкласа (*superclass*) док изведену класу називамо подкласа (*subclass*). Кључна (резервисана) реч `extends` се користи да означи да једна класа наслеђује другу класу.

Општи облик дефиниције наслеђивања класа је:

```
/* nadklasa klasa */
class Nadklasa{
    ...
}

/* podklasa nasledjuje nadklasu */
class Podklasa extends Nadklasa {
    ...
}
```

Подкласа у Јави може да наследи само једну надкласу. Јава не дозвољава вишеструко наслеђивање класа.

Уколико се кључна реч `final` користи код дефинисања класе, онемогућава се наслеђивања таквих класа.

```
final class A{
    ...
}

/* klasa B ne moze da nasledi klasu A jer je klasa A final. */
class B extends A {
    ...
}
```

Уколико се кључна реч `final` користи код дефиниције методе, онемогућава се прекривање таквих метода у изведеним класама.

Пример дефинисања `final` методе :

```
abstract class AbstCB{
    public final void m1(){
        System.out.println("Ova metode ne moze da se prekrije!");
    }

    public abstract void m2();
}

class CB extends AbstCB{
    /* ova metodA ne moze da se prekrije */
    public void m1(){
        System.out.println("Ova metode ne moze da se prekrije!");
    }
    public void m2(){
        System.out.println("Prekrivena abstraktna metoda m2()");
    }
}
```

Кључна реч `super`

У Јави кључна реч `super` се користи да укаже на класу која је на првом вишем нивоу хијерархије од класе, која у оквиру неке од својих метода или конструктора користи кључну реч `super`.

Постоје два случаја коришћења кључне речи `super`:

- а) када се жели приступити чланицама надређене класе
- б) када се жели приступити конструктору надређене класе

Пример коришћења кључне речи `super`:

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class Nasledjivanje2{

    public static void main(String[] args){
        CA obja = new CA(2,3);
        obja.prikazi();
        CB objb = new CB(2,3,5);
        objb.prikazi();
    }
}
```

```

class CA{
    int atr_a1;
    int atr_a2;

    CA(int a1,int a2){
        atr_a1 = a1;
        atr_a2 = a2;
    }
    void prikazi(){
        System.out.println(atr_a1 + ", " + atr_a2);
    }
}
class CB extends CA{
    int atr_b1;
    CB(int a1,int a2, int b1){
        /* poziv konstruktora nadklase */
        super(a1,a2);
        atr_b1 = b1;
    }
    /* prekrivena metoda prikazi() */
    void prikazi(){
        /* poziv metode nadklase */
        super.prikazi();
        System.out.println(atr_b1);
    }
}

```

У овом примеру коришћена је кључна реч `super` у класи `CB` и то у:

- конструктору класе
- прекривеној методи `prikazi()`.

У конструктору класе `CB` кључна реч `super` се користи за позив параметризованог конструктора надкласе `CA`. Кључна реч `super` у телу конструктора мора да буде прва наредба.

У прекривеној методи `prikazi` класе `CB` кључна реч `super` се користи како би се позвала истоимене метода надкласе.

Компабилност објектних типова

Компабилност објектних типова омогућава да објекат основне класе може да добије референцу било ког објекта који је изведен из те класе.

```

/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class Nasledjivanje2{
    public static void main(String[] args){
        CA obja = new CA(2,3);
        CA obja1;
        CB objb = new CB(2,3,5);
        obja1 = obja;
        obja1.prikazi();
        obja1 = objb; // kompatibilnost objektnih tipova
        obja1.prikazi(); // Poziva metodu prikazi klase CB
    }
}

class CA{
    int atr_a1;
    int atr_a2;
    CA(int a1,int a2){
        atr_a1 = a1;
        atr_a2 = a2;
    }
    void prikazi(){
        System.out.println(atr_a1 + ", " + atr_a2);
    }
}

```

```

class CB extends CA{
    int atr_b1;
    CB(int a1,int a2, int b1){
        /* poziv konstruktora nadklase */
        super(a1,a2);
        atr_b1 = b1;
    }
    /* prekrivena metoda prikazi() */
    void prikazi(){
        /* poziv metode nadklase */
        super.prikazi();
        System.out.println(atr_b1);
    }
}

```

Јак полиморфизам

У објекто-оријентисаним програмским језицима разликујемо обичне и виртуелне методе. Обичне методе се повезују са програмом у време компајлирања (*early binding*), док се виртуелене методе повезују са програмом у време извршења програма (*late binding*), помоћу табела виртуелних метода. У Јави су све методе подразумевано виртуелне, што значи да се оне повезују са програмом у време извршења програма. Касно повезивање метода заједно са концептот наслеђивања и концептот компатибилности објектних типова омогућава јак полиморфизам.

Задаци за вежбање:

Задатак ОК3: Објаснити концепт наслеђивања на сопственом примеру.

Задатак ОК4: Дефинисати низ од 10 елемента при чему елементи низа могу бити објекти класе `Student` и-или објекти класе `Radnik`. Напунити низ објектима ових класа а потом приказати само студенте.

Задатак ОК5: Дефинисати низ у коме се чувају и цели и реални бројеви. Написати методу која ће проверити да ли у низу има више целих или реалних бројева.

Питања :

1. Објаснити разлику између преклапања и прекривања метода и дати пример програмског кода
2. Објаснити концепт компатибилности објектних типова и дати пример програмског кода.
3. Да ли Јава класа може да наследи више других Јава класа?
4. Објаснити пример коришћења кључне речи `final` испред назива класе и методе.

1.2.3 Апстрактне класе

Апстрактне класе се дефинишу на исти начин као и обичне класе, али оне за разлику од обичних класа не могу имати појављивања. Апстрактне класе се јављају у случају постојања потребе да класа има бар једну апстрактну методу¹⁰. Апстрактна метода нема тело и одговорност реализације те методе се преноси до методе класе која је наследила апстрактну класу.

Испред апстрактне класе и апстрактне методе се ставља кључна реч `abstract`.

Пример дефинисања апстрактне класе:

```

/* apstraktna klasa AbstCA */
abstract class AbstCA{
    public void m1(){
        System.out.println("Metoda m1() apstraktne klase AbstCA.");
    }
}

```

¹⁰ Апстрактне класе не морају да имају ниједну апстрактну методу.

У овом примеру дефинисана је апстрактна класа `AbstCA` која не садржи ни једну апстрактну методу.

Пример дефинисања апстрактне класе која садржи једну апстрактну методу:

```
/* apstraktna klasa AbstCA */
abstract class AbstCA{
    public void m1(){
        System.out.println("Metoda m1() apstraktne klase AbstCA.");
    }
    public abstract int m2(int a,int b);
}
```

Апстрактна метода нема тело и одговорност реализације те методе се преноси до методе класе која је наследила апстрактну класу.

```
/* konkretna klasa CA koja nasledjuje apst.klasu*/
class CA extends AbstCA{
    /* realizacija metode m2 koja je apstraktna u klasi AbstCA*/
    public int m2(int a,int b){
        return a-b;
    }
}
```

У овом примеру дефинисана је класа `CA` која прекрива апстрактну методу `m2` класе `AbstCA`.

Задаци за вежбање:

Задатак ОК6: Ниже је дат део програмског кода који је потребно допунити како би се на екрану приказала површина круга чији је полупречник 12.7мм.

```
abstract class GTelo{
    abstract double izracunajPovrsinu();
}
class Krug{

}
class Glavna{
    public static void main(String[] arg) {
        Krug k = new Krug();
        k.postaviPoluprecnik(12.7);

        GTelo gt;
        gt = k;
        double p = gt.izracunajPovrsinu();
        System.out.println("Povrsina je: "+p);
    }
}
```

Питања :

- Објаснити разлику између апстрактних и обичних класа. Дати пример програмског кода.

1.2.4 Интерфејси

Концепт интерфејса

Интерфејс је концепт који раздваја спецификацију метода од њихове имплементације. Све методе у интерфејсу су апстрактне методе.

Пример дефинисања интерфејса:

```
interface Ia{
    void m1();
    int m2(int a, int b);
}
```

У интерфејсу се даје спецификација метода (тј. методе се именују и дефинише се њихов потпис), док се имплементација метода из интерфејса врши у класама које тај интерфејс имплементирају (наслеђују).

У примеру који следи дата је класа `Ca` која имплементира интерфејс `Ia` из претходног примера.

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class Ca implements Ia{
    /* prekrivena metoda m1() iz interfejsa */
    public void m1(){
        System.out.println("Prekrivena metoda m1() klase Ca.");
    }
    /* prekrivena metoda m2() iz interfejsa */
    public int m2(int a, int b){
        return a-b;
    }
    /* metoda klase m3() */
    public void m3(){
        System.out.println("Metoda m3() klase Ca.");
    }
}
```

Поређење интерфејса и апстрактне класе

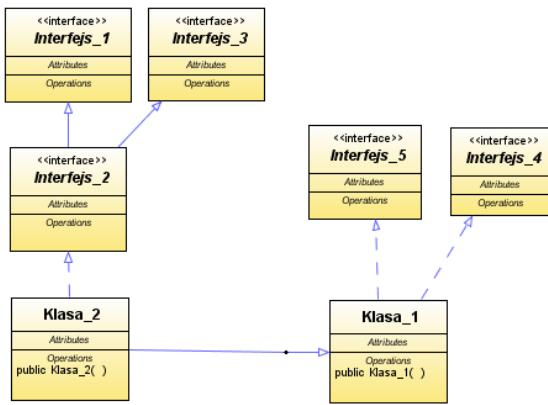
- Код интерфејса се могу дефинисати само константе (не и атрибути). Апстрактна класа може да има атрибуте.
- Свака метода у интерфејсу има само потпис без имплементације. Апстрактна класа може да има конкретне методе.
- Пошто су све методе дефинисане у интерфејсу апстрактне методе, испред имена методе се не стави кључна реч `abstract`, док се код апстрактних метода апстрактних класа ставља кључна реч `abstract`, јер оне могу да имају и конкретне методе.
- Метода класе која реализује методу из интерфејса мора бити јавна (`public`).
- Ни интерфејс ни апстрактна класе не могу да имају своја појављивања.

Правила наслеђивања у Јави

Постоје следећа правила код наслеђивања:

- Интерфејс може да наследи више интерфејса.
- Класа може да наследи више интерфејса.
- Класа не може да наследи више класа. Класа може да наследи само једну класу.
- Интерфејс не може да наследи класу.
- Класа може у исто време да наследи класу и да имплементира један или више интерфејса.

На дијаграму класа (Слика ДКН) приказана су ова правила.



Слика ДКН. Дијаграм класа правила наслеђивања у Јави

Задаци за вежбање:

Задатак ОК7: Ниже је дат програмски код који треба допунити како би се на екрану приказао износ динара за унету вредност стране валуте (долара) .

```

interface IMenjacnica{
    double kurs_dolar = 62.50;
    double dolarToDinar(int dolar);
}

class CMenjacnica implements IMenjacnica{

}

class Menjacnica{
    public static void main(String[] args){
        CMenjacnica obj = new CMenjacnica();
    }
}
  
```

Питања :

- Објаснити разлику између Јава интерфејса и Јава класе.

1.2.5 Изузети

Изузетак представља случај који се дешава у току извршења програма који може да поремети нормалан рад програма.

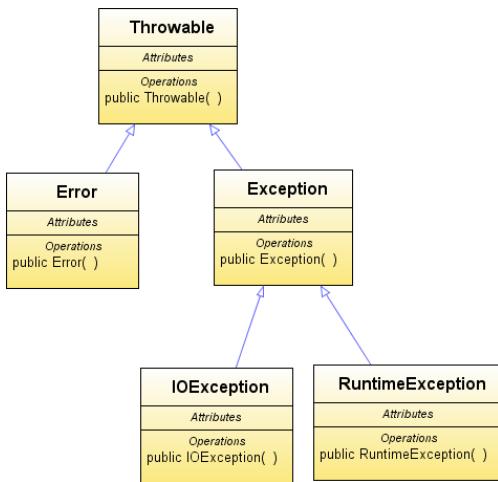
Изузетке могу генерисати:

- методе постојећих Јавиних класа или
- методе корисничких класа.

Када се деси нека грешка у програму, метода у којој се десила грешка прави (баца) изузетак који се односи на ту грешку.

Изузетак се затим хвата (*catch*) и обрађује. Места у програму где се хвата грешка, називају се тачке пресретања грешака (*error-trapping*).

Изузетак представља објекат класе `Throwable` или класе која је изведена из класе `Throwable`. Јавине класе које омогућавају рад са изузетцима су приказане на слици ЈКИ.



Слика ЈКИ: Јавине класе за рад са изузетима

У примеру који следи дат је Јава програмски код у коме нису обрађени изузетци.

```

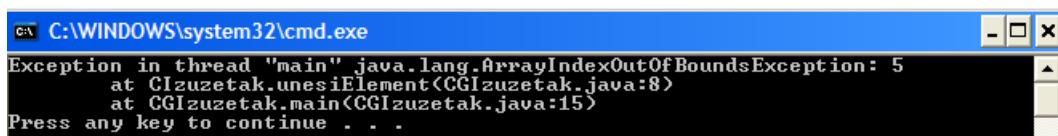
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class CGIzuzetak{
    public static void main(String[] args){
        CIzuzetak obj = new CIzuzetak();
        obj.unesiElement(5,10);
    }
}
  
```

```

class CIzuzetak{
    int []x;
    CIzuzetak(){
        x = new int[5];
    }

    void unesiElement(int pozicija, int element){
        x[pozicija] = element;
    }
}
  
```

У овом примеру дефинисана је класа `CIzuzetak` која садржи методу `unesiElement` која убацују нови елемент у низ на задату позицију. Проблем настаје при позиву ове методе у случају када је димензија низа мања од позиције на коју се убацује нови елемент. У том случају покреће се Јавин стандардни обрађивач грешке и приказује опис грешке као и стек позива метода пре него што се изузак десио. Опис грешке приказан је на слици ОГ.



Слика ОГ: Опис грешке

Овај проблем се може решити на 2 начина:

Први пример решења проблема:

```
void unesiElement(int pozicija, int element){
    try{
        x[pozicija] = element;
    }catch(ArrayIndexOutOfBoundsException aiobe ){
        System.out.println("Pozicija nije dobro uneta!");
    }
}
```

У овом примеру се очекује грешка при извршењу Јава наредбе

```
x[pozicija] = element;
```

Ова наредба се ставља унутар `try - catch` блока.

Други пример решења проблема:

```
class CIIzuzetak{
    public static void main(String[] args){
        CIzuzetak obj = new CIzuzetak();
        try{
            obj.unesiElement(5,10);
        }catch(Exception e){
            System.out.println("Greska pri pozivu metode!");
        }
    }
}
```

У овом примеру се очекује грешка при извршењу Јава методе `unesiElement`

```
obj.unesiElement(5,10);
```

И ова наредба се такође ставља унутар `try-catch` блока.

Уколико постоји потреба могуће је направити изузетак коришћењем наредбе `throw`.

Пример креирања изузетка у Јави.

```
void unesiElement(int pozicija, int element) throws Exception{
    if (element % 2 == 0)
        throw new Exception("Nije dozvoljen unos parnih projeva");
    x[pozicija] = element;
}
```

У овом примеру унутар методе `unesiElement` се креира изузетак наредбом `throw new Exception()`. Креираном изузетку се као параметар прослеђује опис грешке.

Уколико се не жели обрада изузетка унутар методе у којој се тај изузетак креирао потребно је објавити компајлеру (обавестити компајлер) да тај изузетак неће бити обрађен и да ће одговорност обраде бити пренета на методу која је изнад у хијерархији. То се постиже помоћу кључне речи `throws` иза које се наводи класа изузетка који неће бити обрађен:

```
void unesiElement(int pozicija, int element) throws Exception
```

Могуће је направити и сопствене класе изузетака. Ове класе мора да наследе класу `Exception`.

Ниже је дат комплетан програмски код у коме је креирана сопствена класа изузетка.

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
```

```
class CKDefIzuzetak{

    public static void main(String[] args){
        CIzuzetak obj = new CIzuzetak();
        try{
            obj.unesiElement(5,10);
        }catch(KorisnickiDefinisanIzuzzetak ke){
            /* prilikom stampanja objekta poziva se metoda toString() */
            System.out.println(ke);
        }
    }
}
```

```
class CIzuzetak{
    int []x;
    CIzuzetak(){
        x = new int[5];
    }

    void unesiElement(int pozicija, int element) throws KorisnickiDefinisanIzuzzetak{
        if (element % 2 == 0) throw new KorisnickiDefinisanIzuzzetak(0);
        x[posicija] = element;
    }
}
```

```
class KorisnickiDefinisanIzuzzetak extends Exception{
    /* ovaj atribut predstavlja
       broj koji ukazuje na gresku koja se desila
    */
    int kodGreske;

    KorisnickiDefinisanIzuzzetak(int kodGreske){
        this.kodGreske = kodGreske;
    }

    /* ova metoda se poziva prilikom stampanja objekta ove klase */
    public String toString(){
        if (kodGreske==0) return "Nije dozvoljeno da broj bude paran!";
        return "";
    }
}
```

Задаци за вежбање:

Задатак ОК8: Написати сопствени пример који показује начин коришћења **throws** кључне речи.

Задатак ОК9: Написати сопствени пример који показује начин коришћења **try-catch** блока.

Питања:

1. Објаснити разлику између коришћења кључне речи **throws** и **throw**.

1.2.6 Пакети

Пакет представља организациону јединицу која може да садржи класе и друге пакете. Концепт пакета омогућава да се у оквиру једног Јава програма може користити две или више истоимених класа које се налазе у различитим пакетима. На тај начин је омогућен полиморфизам класа.

Назив пакета је директно повезан са системом директоријума текућег диска. То практично значи да име пакета треба да одговара имену директоријума у коме се налазе датотеке везане за тај пакет.

Један пакет може бити везан за више датотека у којима се налазе класе. То практично значи да више класа може бити везано за један пакет.

Пример дефинисања класе унутар пакета:

```
package p1;
class GlavnaP1{
    public static void main(String[] args){
        C1 obj1 = new C1();
        obj1.c1m1();
    }
}
```

```
package p1;
class C2{
    int c2m1(){
        System.out.println("пакет p1, класа C2, метода c2m1().");
        return 0;
    }
}
```

```
package p1;
class C1{
    void c1m1(){
        System.out.println("пакет p1, класа C1, метода c1m1().");
        C2 obj2 = new C2();
        obj2.c2m1();
    }
}
```

За овај пример направљан је директоријум p1. У директоријуму p1 налазе се датотеке GlavnaP1.java, C2.java, C1.java. У овом примеру све класе (GlavnaP1, C2, C1) налазе се унутра једног истог пакета p1. (Слика ПК1)

```
C:\P1
C1.java
C2.java
GlavnaP1.java
```

Слика ПК1: Структура директоријума

Приликом превођења (*compiling*) Јава програма чије су класе организоване унутар пакета потребно је се позиционирати изнад почетног (*root*) пакета и онда позвати Јава преводилац (*compiler*). Исто је потребно урадити и приликом покретања Јава програма. Ниже су наведене наредбе за превођење Јава класа из предходног примера:

```
C:\>javac p1/C2.java
C:\>javac p1/c1.java
C:\>javac p1/GlavnaP1.java
```

Из овог примера може се закључити да се приликом превођења Јава класе која се налази унутар неког пакета испред назива класе наводи име пакета. Пуно име класе чини име пакета у коме се та класа налази и име саме класе.

Уколико се у Јава програму користе класе из других пакета неопходно је да се те класе увезу. У следећем примеру класе из претходног примера распоређене су унутар различитих пакета и то:

- класа GlavnaP1 се налази унутар пакета p1
- класа C2 се налази унутар пакета p1.p2 који је подпакет пакета p1
- класа C1 се налази унутар пакета p1.p1 који је подпакет пакета p1

На слици ПК2 приказан је систем директоријума за овај пример.

```
C:\p1
    GlavnaP1.java
    |
    +-- p1.p1
        C1.java
    |
    +-- p1.p2
        C2.java
```

Слика ПК2: Систем директоријума

Ниже је приказан комплетан програмски код за овај пример.

```
package p11;
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
/* uvozi se klasa C1 koja se nalzi u paket p11.p11p1 */
import p11.p11p1.C1;
class GlavnaP1{
    public static void main(String[] args){
        C1 obj1 = new C1();
        obj1.c1m1();
    }
}
```

```
package p11.p11p2;
public class C2{
    public int c2m1(){
        System.out.println("paket p11.p11p2, clasa C2, metoda c2m1().");
        return 0;
    }
}
```

```
package p11.p11p1;
/* uvozi se klasa C2 koja se nalzi u paket p11.p11p2 */
import p11.p11p2.C2;
public class C1{
    public void c1m1(){
        System.out.println("paket p11.p11p1, clasa C1, metoda c1m1().");
        C2 obj2 = new C2();
        obj2.c2m1();
    }
}
```

Приликом увожења класе из неког пакета наводи се пуно име пакета. Име пакета почиње од почетног (*root*) пакета, а завршава се пакетом у коме се налази класа која се увози.

У овом примеру изменењен је ниво приступа до класа и метода. Постављен је јаван (*public*) начин приступа за све класе и методе, како би се ове класе и методе користиле унутар других пакета.

1.2.7 Улазно-излазни токови

У Јави објекат из кога се чита секвенца бајтова назива се **улазни ток** (*input stream*), док се објекат у кога се уписује секвенца бајтова назива **излазни ток** (*output stream*).

Улазни и излазни токови, се обрађују на исти начин, независно од тога да ли су везани за монитору, екран, датотеку или мрежу.

У Јави постоје бинарни и знаковни токови. Велики број класа које се односе на токове (бинарне улазне и излазне, знаковне улазне и излазне) налазе се у пакету `java.io`.

Треба нагласити да се на најнижем нивоу обраде улазно-излазних операција ради са бинарним подацима. То значи да рад са знаковним подацима у *unicode* формату подразумева њихову конверзију у бинарни формат и обратно, у зависности од тога да ли се ради са излазним или улазним током.

У примерима ОК-10 и ОК-11 за унос података са тастатуре се користи знаковни ток BufferedReader. Пошто на најнижем ниво Јава ради са бинарним токовима, онда је потребно да се бинарни ток конвертује у знаковни ток а то се чине преко класе InputStreamReader.

Пример ОК-10: Креирати класу ZnakovniTokTastatura која садржи методу citajSaTastature(). Метода треба преко повратног типа да врати унети податак као String.

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
import java.io.*;
class GlavnaZT{
    public static void main(String[] args){
        ZnakovniTokTastatura ztt = new ZnakovniTokTastatura();
        try{
            String ulaz = ztt.citajSaTastature();
            System.out.println("Uneti podatak: "+ulaz);
        }catch(IOException ioe){
            System.out.println(ioe);
        }
    }
}
```

```
import java.io.*;
class ZnakovniTokTastatura{
    public String citajSaTastature() throws IOException{
        BufferedReader inTastatura = new BufferedReader (
                new InputStreamReader(System.in));
        String ulaz = inTastatura.readLine();
        return ulaz;
    }
}
```

Пример ОК-11: Проширити главни програм из примера UI-1 тако што корисник треба да добије информацију да ли је унет цео број са тастатуре или не.

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
import java.io.*;
class GlavnaZT{
    public static void main(String[] args){
        ZnakovniTokTastatura ztt = new ZnakovniTokTastatura();
        try{
            String ulaz = ztt.citajSaTastature();
            int broj = Integer.parseInt(ulaz);
            System.out.println("Uneti podatak je broj");
        }catch(IOException ioe){
            System.out.println(ioe);
        }catch(NumberFormatException nfe){
            System.out.println("Nije unet broj");
        }
    }
}
```

У овом примеру метода Integer.parseInt се користи за конверзију податка типа String у тип податка int.

Уписивање знаковних података у датотеке

`FileWriter` класа представља улазни знаковни ток који се односи на датотеку. У примеру OK-12 користи се класа `PrintWriter` и њена метода `println` за упис податка у датотеку.

Пример OK-12:

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
class GlavnaZTDat{
    public static void main(String[] args){
        ZnakovniTok objZt = new ZnakovniTok();
        try{
            objZt.upisiUdatoteku();
        }catch(IOException ioe){
            System.out.println(ioe);
        }
    }
}
```

```
class ZnakovniTok{
    public void upisiUdatoteku() throws IOException{
        FileWriter outDatoteka = new FileWriter("izlaz2.txt");
        PrintWriter poutDatoteka= new PrintWriter(outDatoteka);
        poutDatoteka.println("prva linija");
        poutDatoteka.println("druga linija");
        poutDatoteka.println("treca linija");
        poutDatoteka.close();
        outDatoteka.close();
    }
}
```

Читање знаковних података из датотеке

`FileReader` класа представља излазни знаковни ток који се односи на датотеку. У примеру OK-13 користи се класа `BufferedReader` и њена метода `readLine` за читање података из датотеке.

Пример OK-13:

Проширити програмски код из примера OK-12 са методом за читање података из датотеке.

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
import java.io.FileReader;
import java.io.BufferedReader;
class GlavnaZTDat{
    public static void main(String[] args){
        ZnakovniTok objZt = new ZnakovniTok();
        try{
            objZt.upisiUdatoteku();

            objZt.procitajIzDatoteke();
        }catch(IOException ioe){
            System.out.println(ioe);
        }
    }
}
```

```

class ZnakovniTok{
    public void upisiUdatoteku() throws IOException{
        FileWriter outDatoteka = new FileWriter("izlaz2.txt");
        PrintWriter poutDatoteka= new PrintWriter(outDatoteka);
        poutDatoteka.println("prva linija");
        poutDatoteka.println("druga linija");
        poutDatoteka.println("treca linija");
        poutDatoteka.close();
        outDatoteka.close();
    }
    public void procitajIzDatoteke() throws IOException{
        FileReader inDatoteka = new FileReader ("izlaz2.txt");
        BufferedReader buffInDatoteka= new BufferedReader (inDatoteka);
        String s = buffInDatoteka.readLine();
        System.out.println(s);
        s = buffInDatoteka.readLine();
        System.out.println(s);
        s = buffInDatoteka.readLine();
        System.out.println(s);
        buffInDatoteka.close();
        inDatoteka.close();
    }
}

```

Серијализација објеката

Серијализација је процес којим се омогућава памћење објеката са њиховим стањем у излазној датотеци. Помоћу серијализације везе између два или више објеката, која је остварена референцама у току извршења програма, бива сачувана у датотеци. То значи да се при поновном читању објеката из датотеке везе између њих обнављају. У суштини сваки објекат када се памти у датотеци добија свој јединствени број. Када се памте везе између објеката, не памте се адресе где се налазе ти објекти, већ се те адресе конвертују у јединствене бројеве објеката који се чувају у датотеци. Касније када се читају објекти, ти јединствени бројеви објеката се инверзним поступком конвертују у адресе, где ће објекти бити сачувани. При томе се остварују, као што смо већ рекли, везе између објеката које су биле пре њиховог памћења у датотеци. Процес читања серијализованих објеката из датотеке и њихово пребацање у објекте програма назива се десеријализација.

Пример ОК-14: Написати програм у Јави који омогућава да се објекат класе *Prijava* чува у датотеци. На пријави се уносе следећи подаци: број индекса студента, шифра предмета и оцена. Сачувати објекат класе *Prijava* у датотеку, а након тога прочитати објекат из датотеке и приказати на екрану име и презиме студента као и назив предмета.

```

/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
import java.io.*;
class GSerijalizacija{
    public static void main(String[] args){
        Student s = new Student("1/00","Laza Lazic");
        Predmet p = new Predmet(1,"Predmet 1");
        Prijava prijava = new Prijava(s,p,8);
        Serijalizacija objSerijalizacija = new Serijalizacija();
        try{
            objSerijalizacija.upisi(prijava);
            Prijava outPrijava = objSerijalizacija.procitaj();
            System.out.println(outPrijava);
        }catch(Exception ex){
            System.out.println(ex);
        }
    }
}

```

```

class Serijalizacija{

    public void upisi(Prijava p) throws Exception{
        FileOutputStream outDat = new FileOutputStream("sdat.txt");
        ObjectOutputStream outSerijDat= new ObjectOutputStream(outDat);
        outSerijDat.writeObject(p);
        outSerijDat.close();
        outDat.close();
    }

    public Prijava procitaj()throws Exception{
        FileInputStream inDat = new FileInputStream("sdat.txt");
        ObjectInputStream inSerijDat= new ObjectInputStream(inDat);
        Prijava p = (Prijava)inSerijDat.readObject();
        return p;
    }
}

```

```

class Prijava implements Serializable{
    Student s;
    Predmet p;
    int ocena;

    Prijava(Student s, Predmet p, int ocena){
        this.s = s;
        this.p = p;
        this.ocena = ocena;
    }
    public String toString(){
        return s.vratiImePrezime() + ", " + p.vratiNaziv() + ", " + ocena;
    }
}

```

```

class Student implements Serializable{
    String brInd;
    String imePrezime;
    Student(String brInd, String imePrezime){
        this.brInd = brInd;
        this.imePrezime = imePrezime;
    }
    public String vratiImePrezime(){
        return imePrezime;
    }
}

```

```

class Predmet implements Serializable{
    int sifra;
    String naziv;

    Predmet(int sifra, String naziv){
        this.sifra = sifra;
        this.naziv = naziv;
    }
    public String vratiNaziv(){
        return naziv;
    }
}

```

Задаци :

Задатак ОК10: Напистати програм који рачуна збир два броја. Обезбедити контролу уноса при чему корисник има 3 покушаја за унос сваког од сабирака. Уколико корисник не унесе лепо бројеве генерисати изузетак који ће бити обрађен у главном програму и приказати поруку: *Подаци нису коректни.*

Задатак ОК11: Направити програм који са улаза прихвата `String`, проверава да ли у унетом `String`-у постоји цифра. Уколико постоји цифра генерисати изузетак: *String није лепо унет.* Уколико у `String`-у не постоји цифра приказати унети `String`.

Задатак ОК12: Напунити низ целих бројева. Све елементе који се појављују 2 или више пута у низу пребацити у датотеку, а потом приказати садржај датотеке.

Задатак ОК13: Напунити датотеку елементима целобројног типа. Пребацити парне елементе датотеке у низ. Уколико постоји више елемената датотеке који треба да се пребаце у низ него што је димензија низа генерисати изузетак са поруком: *Нису сви елементи из датотеке пребачени у низ.* Приказати све елементе низа.

Задатак ОК14: Проширити задатак из примера ОК-14 при чему треба омогућити кориснику да унесе п објекта класе `Prijava`, као и да након тога прикаже садржај датотеке.

2.НИТИ

Едсгер Дијкстра је рекао да се “конкурентност дешава када постоје два или више извршних токова (процеса) који су способни да се извршавају симултano (истовремено)”. Процеси могу истовремено да користе дељене ресурсе (*shared resources*) што може да резултује непредвиђеним понашањем система. Увођење **међусобног искључења (mutual exclusion)** може да спречи непредвиђено понашање код коришћења дељених ресурса али може да доведе до појаве **мртвог закључавање (deadlock)** и **гладовања (starvation)**.

Deadlock је мултитаскинг проблем¹¹ који се дешава када два процеса заузму ресурсе и медусобно се цекају да ослободе те ресурсе. **Starvation** је мултитаскинг проблем који се дешава када неки процес заузме неки ресурс и не дозвољава другим процесима да га користе. То доводи до тога да други процеси не могу да се до краја изврше.

Уколико више процеса међусобно сарађују у извршењу неког задатка јавља се проблем њихове међусобне комуникације и размене података јер сваки процес заузима посебан меморијски простор. Тај проблем је решен појавом **нити (threads)** које деле исти меморијски простор.

Јава је један од програмских језика који подржава вишенитно програмирање. То значи да један програм (процес) може да обавља више нити истовремено (конкурентно). **Нит представља део програма који може истовремено да се извршава са другим нитима истог програма.**

Нити, као што је речено, деле исти адресни простор у оквиру једног процеса и комуникација између њих је доста једноставнија у односу на комуникацију између процеса. Радом са више процеса управља оперативни систем, док радом са више нити управља Јава окружење.

2.1 Главна програмска нит

Када програм у Јави почне да се извршава, он аутоматски креира и извршава једну нит која се зове **главна програмска нит**.

```
/*
 * Primer NT1: Napisati program koji ce da ukaze na glavnu nit koju treba
 * uspavati 5 sekundi.
 */

/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class NT1 {

    public static void main(String args[]) {
        Thread gnit = Thread.currentThread();
```

¹¹ *Multitasking* је метода помоћу које више задатака (*tasks*), односно процеса, деле заједничке ресурсе као што је *CPU* (*Central Processing Unit*). У случају компјутера са једним *CPU*, у неком тренутку времена само један задатак може да се извршава, што практично значи да *CPU* активно извршава само инструкције тог задатка. *Multitasking* омогућава да се у неком периоду времена више процеса извршава, тако што прави распоред (*scheduling*) који задатак ће се извршавати у ком тренутку времена а који задаци ће чекати док на њих не дође ред. Додељивање *CPU* од једног до другог задатка се назива *контекстна додела (context switch)*. Када се *context switch* дешава учестало ствара се илузија истовремености извршења више задатака. *Multitasking* омогућава да се изврши више задатака на једном *CPU* у односу на компјутере са више *CPU* (мултипроцесорски компјутери) који не користе *multitasking*.

```

System.out.println("Glavna nit:" + gnit + '\n' + "Pauza od 5 sekundi");
try {
    gnit.sleep(5000); // 5 sekundi pauze
} catch (InterruptedException e) {
    System.out.println("Prekid niti");
}
gnit.setName("Glavna"); // Promena naziva niti
System.out.println("Glavna nit:" + gnit + '\n' + "Naziv glavne niti:" + gnit.getName());
}
/*
Rezultat:
Glavna nit:Thread[main,5,main]
Pauza od 5 sekundi.
Glavna nit:Thread[Glavna,5,main]
Naziv glavne niti: Glavna
*/

```

2.2 Прављење нити

У Јави се нит може направити на 2 начина:

- Реализацијом интерфејса `Runnable`
- Проширењем класе `Thread`

2.2.1 Прављење нити реализацијом класе `Runnable`

Када се нит прави реализацијом интерфејса `Runnable`, тада је једино потребно се имплементира (реализује) метода `run()` интерфејса `Runnable`¹².

```

/* Primer NT2: Napraviti nit pomocu interfejsa Runnable.*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */

class NT2 implements Runnable {
    NT2() {
        //nova nit ce pozvati run metodu od this objekta
        nit = new Thread(this, "Nova nit");
        nit.start();
    }
    public void run() {
        System.out.println("Nit:" + nit);
    }

    public static void main(String args[]) {
        NT2 nn = new NT2();
        Thread gnit = Thread.currentThread();
        gnit.setName("Glavna nit");
        System.out.println("Nit:" + gnit);
    }
    Thread nit;
}
/*
Rezultat:
Nit:Thread[Glavna nit,5,main]
Nit:Thread[Nova nit,5,main]
*/

```

Упрошћено објашњење односа између класа `Thread` и `NT2`

Однос између класа `NT2` и `Thread`, односно њихових објеката, показаћемо на примеру који наглашава најважније аспекте ове комуникације. Треба нагласити да је класа

¹² `main()` метода се за главну нит у суштини понаша исто као и `run()` метода за нити које су креиране у `main()` методи.

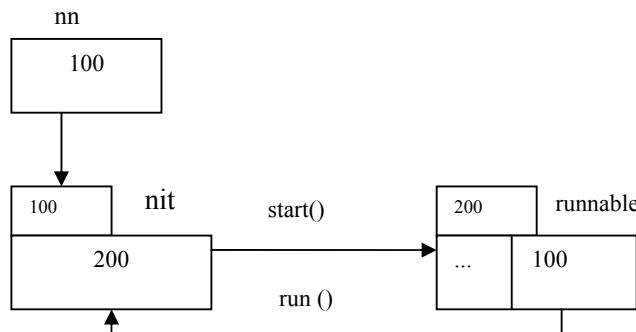
Thread која је дата у овом примеру представља поједностављену верзију оригиналне Јавине класе Thread.

```
/*
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class NT2 implements Runnable {
    NT2() {
        nit = new Thread(this, "Nova nit");
        nit.start();
    }
    public void run() {
        ...
    }
    public static void main(String args[]) {
        NT2 nn = new NT2();
        ...
    }
    Thread nit;
}

public class Thread implements Runnable {
    ...
    final Runnable runnable;
    ...
    public Thread(Runnable target, String name) {
        this.runnable = target;
    }
    public synchronized void start() {
        ... // Kreiranje i inicijalizacija prirodne niti niti preko operativnog sistema
        runnable.run();
    }
    ...
}
```

Изглед оперативне меморије наведеног примера:

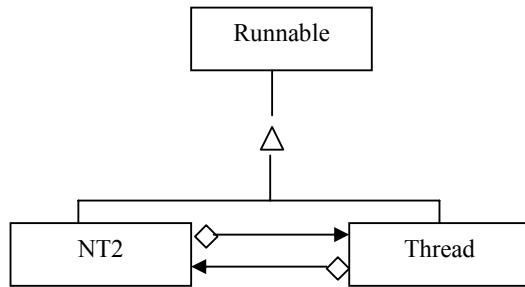


Објекти на које показују nn и nit (који се налазе на адресама 100 и 200) показују један на други. На тај начин је могуће да објекат класе NT2 (на адреси 100) позове методу start() од објекта класе Thread (на адреси 200), односно да објекат класе Thread позове методу run() од објекта класе NT2.

Метода start() класе Thread има задатак да креира¹³ и иницијализује природну нит и да након тога позове методу run() класе NT2.

Дијаграм класа наведеног примера:

¹³ Креирање природне нити (*native thread*) ради оперативни систем. То значи да ће преко методе start() бити позвана нека од метода (операција) оперативног система која је задужена за креирање природне нити. У даљем тексту ће то бити мало детаљније објашњено.



Детаљније објашњење односа између класа Thread и NT2

Да би се детаљније схватио наведени пример¹⁴ навешћемо најважније атрибуте и методе класа **Thread** и **VMThread**¹⁵.

```

public class Thread implements Runnable {
    ...
    final Runnable runnable;
    ...
    public Thread(Runnable target, String name) {
        this(null, target, name, 0);
    }
    ...
    /* Ovaj se konstruktor koristi u primeri NT3 */
    public Thread(String name){
        this(null, null, name, 0);
    }
    public Thread(ThreadGroup group, Runnable target, String name, long size) {
        ...
        this.runnable = target;
        ...
    }
    ...
    public synchronized void start() {
        ...
        VMThread.create(this, stacksize);
    }
    ...
    public void run() {
        if (runnable != null) {
            runnable.run();
        }
    }
    ...
}
  
```

```

final class VMThread {
    volatile Thread thread;
    ...
    private VMThread(Thread thread) {
        this.thread = thread;
    }
    ...
    private void run() {
        ...
        thread.run();
    }
}
  
```

¹⁴ Наведене класе су такође потребне да би се схватио пример NT3, када се креира нит наслеђивањем класе Thread.

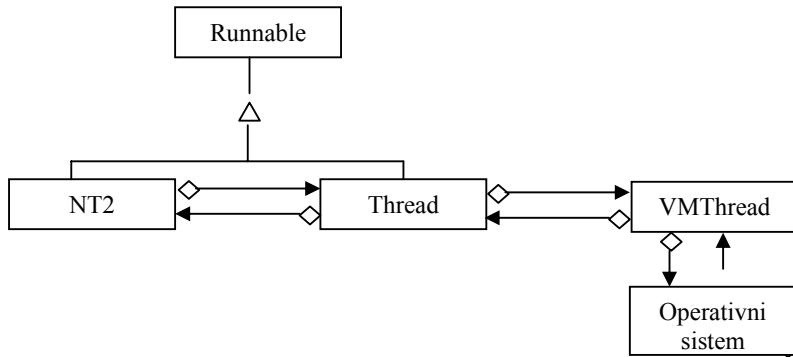
¹⁵ Изворни код наведених класа се може наћи на: <http://www.docjar.com/html/api/java/lang/Thread.java.html> и <http://www.docjar.com/html/api/java/lang/VMThread.java.html>

```

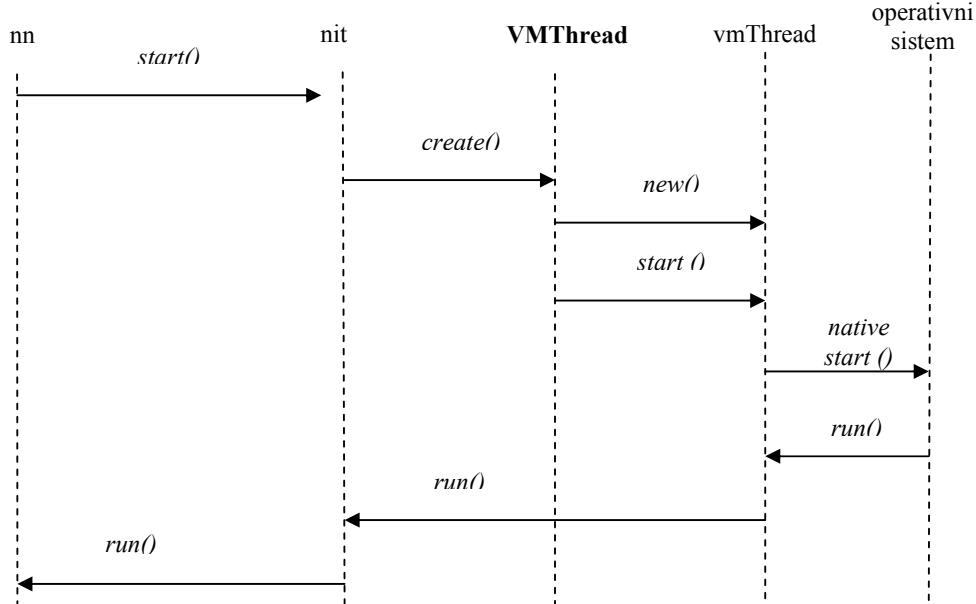
...
    static void create(Thread thread, long stacksize) {
        VMThread vmThread = new VMThread(thread);
        vmThread.start(stacksize);
        ...
    }
    ...
    /* Create a native thread on the underlying platform and start it executing on the run
method of this object.
 * @param stacksize the requested size of the native thread stack
 */
    native void start(long stacksize);
    ...
}

```

Детаљни дијаграм класа наведеног примера:



Секвенцни дијаграм односа између објеката наведеног примера:



На основу наведених дијаграма класа и секвенцног дијаграма може се закључити да позивом методе **start()** објекта нит класе **Thread** започиње процес позива метода (**create**, **new**, **start**) класе **VMThread** који се завршава позивом методе **start()** оперативног система која креира природну нит. Након креирања природне нити оперативни систем позива методу **run()** класе **VMThread** која позива методу **run()** класе **Thread** која на крају позива методу **run** класе **NT2**.

2.2.2 Прављење нити проширењем класе Thread

Када се нит прави проширењем класе Thread, тада је потребно се прекрије метода run() класе Thread. Класа Thread поред методе run() садржи и друге методе које могу да се прекрију, за разлику од интерфејса Runnable који има само методу run().

```
/*Primer NT3: Napraviti nit pomocu klase Thread.*/

/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */

class NT3 extends Thread {
    NT3() {
        super("Nova nit");
        start();
    }
    public void run() {
        System.out.println("Nit:" + currentThread());
    }
    public static void main(String args[]) {
        NT3 nn1 = new NT3();
        Thread gnit = Thread.currentThread();
        gnit.setName("Glavna nit");
        System.out.println("Nit:" + gnit);
    }
}
/*
Rezultat:
Nit: Thread[Glavna nit,5,main]
Nit: Thread[Nova nit,5,main]
*/
```

Задаци:

1. Нацртати изглед детаљног дијаграма класа за пример NT3.
2. Нацртати и објаснити секвенцијни дијаграм за методу start() која се позива у конструктору класе NT3.
3. У методи start() класе Thread код наредбе:
`VMThread.create(this,stacksize);`
шта је **this**?
4. У методи run() класе VMThread:
`private void run() { ... thread.run(); }`
шта позива наредба `thread.run();`?

2.3 Прављење више нити

У једном Јава програму може да постоји више нити.

```
/* Primer NT31: Napraviti u jednom programu vise niti.*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */

class NT31 extends Thread {
    NT31(String nn) {
        super(nn);
        start();
    }
    public void run() {
        System.out.println("Nit:" + currentThread());
    }
}
```

```

public static void main(String args[]) {
    NT31 nn1 = new NT31("Nova nit1");
    NT31 nn2 = new NT31("Nova nit2");
    NT31 nn3 = new NT31("Nova nit3");
    Thread gnit = Thread.currentThread();
    gnit.setName("Glavna nit");
    System.out.println("Nit:" + gnit);
}
}

/*
Rezultat:
Nit: Thread[Glavna nit,5,main]
Nit: Thread[Nova nit1,5,main]
Nit: Thread[Nova nit2,5,main]
Nit: Thread[Nova nit3,5,main]
*/

```

2.4 Станања нити

Нит може бити у једном од 4 станања:

- ново (*new*)
- извршно (*runnable*)
- блокирано (*blocked*)
- свршено (*dead*)

Нова нит

У почетку нит је декларисана: `Thread nit;`

Када се нит креира са `new` оператором: `nit = new Thread(this, "Nova nit");` нит прелази у стање **"ново"**. У том стању, нит се још не извршава. Након тога, позива се метода `start()`.

Извршавање нити

Након позива методе `start()`, нит прелази у стање **"извршно"**. Извршно стање не значи аутоматски да се нит **"извршава"** (*running*). Нит се извршава када контрола програма пређе на тело методе `run()`, коју позива метода `start()`.

```

// NapraviNit.java
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class NT3 implements Runnable{

    NT3() {
        nit = new Thread(this, "Nova nit"); // Nit je u stanju "novo"
        nit.start(); // Nit je u stanju "izvrsno"
    }

    public void run() {
        System.out.println("Nit:" + nit);
    } // Nit je u stanju "izvrsava"

    // Nakon izvrsenja tela metode run() nit prelazi u stanje "svrseno"
    public static void main(String args[]) {
        NT3 nn = new NT3();
        Thread gnit = Thread.currentThread();
        gnit.setName("Glavna nit");
        System.out.println("Nit:" + gnit);
    }
}
```

```

Thread nit;
}
/*
Rezultat:
Nit: Thread[Glavna nit,5,main]
Nit: Thread[Nova nit,5,main]
*/

```

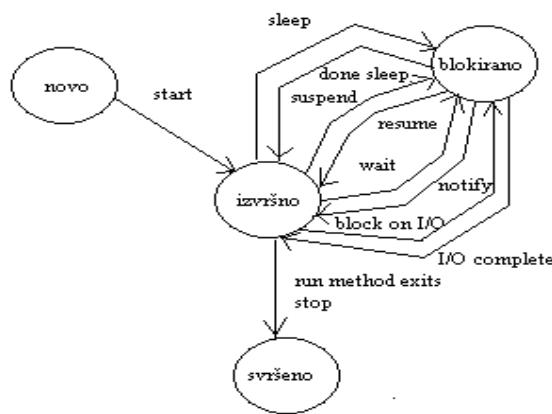
Оперативни систем је задужен да обезбеди процесорско време за нити, како би се оне извршиле. Старије верзије оперативних система (нпр. *Solaris* оперативни систем) су неку нит (почетна нит) извршавале, све до појаве неке нити вишег приоритета. Почетна нит би тада прешла у блокирано стање, док би нит вишег приоритета наставила да се извршава док се не изврши или док се не појави нека нит још већег приоритета.

Код новијих оперативних система (нпр. *Windows 2000* оперативни систем), свака нит добија део процесорског времена (*time – slicing*) да изврши свој задатак. Овај приступ је бољи, јер је у складу са вишенитном извршењем програма код Јаве . На машинама са више процесора, сваки процесор може да извршава једну нит. То значи да више нити може да се извршава истовремено.

Улазак нити у блокирано стање

- Нит прелази у блокирано стање, када се деси једна од следећих акција:
- Нит позива `sleep()` методу. Нит остаје "успавана" задати број милисекунда.
- Нит позива операцију, која је блокирала неки од улазно/излазних уређаја. Таква операција се неће наставити, све док се не ослободи заузети улазно/излазни уређај.
- Низ позива `wait()` методу.
- Нит покушава да закључча објекат који је већ закључан од стране друге нити.
- Нит позива `suspend()` методу. Ова метода је застарела и више се не користи.

На следећој слици се виде стања нити и могуће транзиције:



Излазак нити из блокираног стања

Нит може да пређе из "блокираног" у "извршно" стање, када се деси једна од следећих акција:

- Нит се "пробудила" након што је прошло време "успаваности ", задато методом `sleep()` .
- Операција, која је позвана у току извршења нити, која је блокирала улазно/излазни уређај је завршена.
- Ако је прва нит позвала `wait()` методу, друга нит мора позвати `notify()` или `notifyAll()` методу како би се деблокирала прва нит.

- Када блокирана нит, која чека на закључан објекат, добије контролу над њим, након што је нека нит престала да држи тај објекат закључан.
- Ако је нит била блокирана методом `suspend()`, она се може деблокирати једино позивом методе `resume()`. Метода `resume()` је такође застарела као и метода `suspend()`

Важна напомена: Блокирана нит се може деблокирати једино инверзном операцијом од операције која ју је блокирала (*sleep – done sleeping, suspend – resume, wait – notify, block on I/O - I/O complete*)

Ако се покуша деблокирати нит са неодговарајућом инверзном операцијом јавиће се `IllegalThreadStateException` изузетак.

Уништење нити

Нит прелази у стање "свршено" у следеће 2 ситуације:

- `run()` метода је престала да се извршава природним путем.
- `run()` метода је престала да се извршава јер се у току њеног извршења десио неухваћен изузетак.

Нит се такође може зауставити ако се позове метода `stop()`. Међутим ова метода је застарела и не користи се јер доводи до нестабилног понашања програма.

Уколико се жели испитати да ли је нит тренутно активна (независно од тога да ли се извршава или је блокирана) користи се метода `isAlive()`, која у том случају враћа `true`. Уколико је нит у стању "ново" или "свршено" метода `isAlive()` враћа `false`. Наведена метода ће касније бити детаљније објашњена.

2.5 Прекид нити

Нит се завршава када се тело `run()` методе изврши. Пошто је `stop()` метода превазиђена, користи се логичка контрола за наставак извршења нити:

```
public void run{
    while(signal) {
        izvrsenje niti...
    }
}
```

Докле год `signal` има вредност `true`, нит се извршава. Када `signal` добије вредност `false`, позивом нпр. методе `promeni()` нит престаје да се извршава.

```
public void promeni(){
    signal = false;
}
```

Међутим у случају када је нит у блокираном стању, прекид извршења нити не може остварити на предходни начин. Тада се користи `interrupt()` метода која позива нит која је блокирана. Она ће прекинути блокаду нити генерирањем изузетка `InterruptedException`. Блокада може престати ако је иста настала на основу `sleep()` или `wait()` методе.

Наведена `interrupt()` метода прекида блокаду нити, али не прекида извршење нити. Уколико се жели да прекид нити преко `interrupt()` методе прекине извршење нити, тада се ухваћени изузетак `InterruptedException` обрађује на одговарајући начин.

```
public void run () {
    try {
        while(signal){
            izvrsenje niti...
        }
    } catch(InterruptedException e) {
        // Prekinuta blokada niti koja je nastala na osnovu sleep() ili wait() metode.
    }
    ...
    // izlaz iz run() metode i prekid izvršenja niti.
}
```

Метода `interrupt()` статус нити поставља на `true`.

У случају када `interrupt()` метода позове нит која није блокирана, тада се неће десити изузетак `InterruptedException`. Због тога се код логичке контроле извршења нити позива метода `interrupted()`, која проверава да ли је текућа нит “претрпела” `interrupt()` методу (да ли је статус нити `true`).

Уколико је статус нити `true` треба да се прекине извршење нити:

```
while( !interrupted() && signal){
    izvrsenje niti...
}
```

ИЛИ

Метода `interrupted()` има спољни ефекат (*side effect*), јер стање нити поставља на `false`.

Поред наведених постоји и метода `isInterrupted()` која враћа стање прекида нити али нема спољни ефекат:

```
while( !isInterrupted() && signal){
    izvrsenje niti...
}
```

На основу наведеног може се закључити да постоје неколико сценарија покушаја прекида нити у зависности од тога: а) да ли је нит блокирана или не и б) како се покушава прекинути нит (`interrupt` методом или логичком контролом).

Сценарио прекида нити логичком контролом – нит није блокирана

Наведени сценарио се састоји из следећих корака:

- нит је покренута и њен статус је постављен на `true`.
- позива се метода промени која мења атрибут `signal` на `false`.
- метода `run` престаје да се извршава јер услов у `while` петљи није више задовољен:

```
while(signal){
    izvrsenje niti...
}
```

Сценарио прекида нити `interrupt` методом – нит није блокирана

- нит је покренута и њен статус је постављен на `true`.
- позива се метода `interrupt()`. Пошто нит није блокирана не генерише се изузетак `InterruptedException`. Статус нити се поставља на `true`.
- метода `run` престаје да се извршава јер услов у `while` петљи није више задовољен:
`while(!interrupted() && signal) { izvrsenje niti... }`
- Метода `interrupted()` враћа статус нити и мења статус нити на `false`.

Слично ће се десити уколико је `while` петља дефинисана као:

```
while( !isInterrupted() && signal) { izvrsenje niti... }
```

Метода `isInterrupted()` враћа статус нити али не мења статус нити (статус нити остаје на `true`).

Сценарио прекида нити логичком контролом – нит је блокирана

- нит је покренута и њен статус је постављен на `true`.
- нит се блокира помоћу `sleep()` или `wait()` методе.
- позива се метода промени која мења атрибут сигнал на `false`.

- метода `run()` не престаје да се извршава јер је нит блокирана и контрола програма не долази до `while` наредбе:

```
while(signal) {
    nit je blokirana ...
}
```

Можемо закључити да се наведеним сценариом не прекида извршење нити.

Сценарио прекида нити `interrupt` методом – нит је блокирана

- нит је покренута и њен статус је постављен на `true`.
- нит се блокира помоћу `sleep()` или `wait()` методе.
- позива се метода `interrupt()`. Пошто је нит блокирана генерише се изузетак `InterruptedException`. Статус нити се поставља на `true`.
- метода `run` престаје да се извршава јер је генерисан изузетак:

```
public void run() {
    try {...}
        while(signal){nit je blokirana
        ...
    }
    catch(InterruptedException e) {
    }
...
}
```

Треба обратити пажњу на следећи потенцијални проблем уколико је `try/catch` блок унутар `while` петље:

```
public void run() {
    while (...) {
        try {
            nit je blokirana ...
        } catch(InterruptedException e) {
        }...
    }
}
```

У наведеном случају неће доћи до прекида нити јер ће контрола програма остати у `while` петљи.

У следећа 2 примера су дати примери прекида нити.

```
/*Primer NT4: Pokazati kako se prekida "uspavanost" niti pomocu metode interrupt().*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */
class NT4 extends Thread {
    NT4() {
        start();
    }
    public void run() {
        try {
            sleep(5000);
        } catch (InterruptedException e) {
        }
        System.out.println("Nova nit probudjena!");
    }
    public static void main(String args[]) throws Exception {
        NT4 nn = new NT4();
        Thread gnit = Thread.currentThread();
        gnit.sleep(2000);
        System.out.println("Glavna nit probudjena!");
        nn.interrupt(); // Kada se iskljuci ova naredba nova nit nn je uspavana
        // 5 sekundi, inace se odmah prekida "uspavanost" niti nn.
    }
}
```

```
Rezultat:
Glavna nit probudjena
Nova nit probudjena
*/
```

Други пример је мало сложенији.

```
/*
Primer NT5: Pokazati kako se prekida izvrsenje niti u 3 situacije:
a) Ukoliko se nit izvrsava u while petlji i nit nije uspavana.
b) Ukoliko se nit izvrsava u while petlji i nit je uspavana.
c) Ukoliko se nit izvrsava u while petlji i nit nije uspavana a zelimo je
prekinuti preko interrupt() metode.
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */

class NT5 extends Thread {
    NT5(int opcijal) {
        signal = true;
        opcija = opcijal;
        start();
    }

    public void run() {
        switch (opcija) {
            case 0:
                ObicanPrekidNiti();
                break;
            case 1:
                InterruptKojiNePrekidaIzvrsenjeNiti();
                break;
            case 2:
                InterruptKojiPrekidaIzvrsenjeNiti();
                break;
            case 3:
                InterruptKojiPrekidaNeuspavanuNit();
                break;
        }
    }
}

// Iz glavnog programa promenice se status signal promenljive sto ce dovesti do prekida
// niti.
public void ObicanPrekidNiti() {
    while (signal) {
    }
    System.out.println("Nova nit 0 probudjena!");
}

// Prekid "uspavanosti" niti vratice kontrolu izvrsenja niti u while petlju. Ovde je
// problem u tome sto se try/catch blok nalazi unutar while petlje.
public void InterruptKojiNePrekidaIzvrsenjeNiti() {
    while (signal) {
        try {
            sleep(5000);
        } catch (InterruptedException e) {
        }
    }
    System.out.println("Nova nit 1 probudjena!");
}

// Prekid "uspavanosti" niti prekinuce izvrsenje.niti, jer je try/catch blok izvan
// while petlje a ne unutar nje.
public void InterruptKojiPrekidaIzvrsenjeNiti() {
    try {
        while (signal) {
            sleep(5000);
        }
    } catch (InterruptedException e) {
    }
    System.out.println("Nova nit 2 probudjena!");
}

// Ukoliko se desi da interrupt metoda pozove neuspavanu nit potrebno je izvrsiti
```

```

// dopunska kontrola while petlje pomocu interrupted() metode koja vraca true ukoliko
// je za tekucu nit pozvana interrupt() metoda.
public void InterruptKojiPrekidaNeuspavanuNit() {
    // ** while(!isInterrupted() && signal) { } // ne menja interrupt status niti
    while (!interrupted() && signal) {
        // menja interrupt status niti na false
        System.out.println("Nova nit 3 probudjena!");
        System.out.println("Status prekida : " + isInterrupted());
        // Status prekida bi bio true da se izvrsila naredba **
    }

    public void prekid() {
        signal = false;
    }

    public static void main(String args[]) throws Exception {
        NT5 nn0 = new NT5(0);
        NT5 nn1 = new NT5(1);
        NT5 nn2 = new NT5(2);
        NT5 nn3 = new NT5(3);
        Thread gnit = Thread.currentThread();
        gnit.sleep(2000);
        System.out.println("Glavna nit probudjena!");
        nn0.prekid();
        nn1.interrupt();
        nn2.interrupt();
        nn3.interrupt();
    }
    boolean signal;
    int opcija;
}
/*
Rezultat:
Glavna nit probudjena
Nova nit 2 probudjena
Nova nit 0 probudjena
Nova nit 3 probudjena
Status prekida: false
*/

```

Задаци за вежбање:

```

/*
Primer NTZ1: Napisati 2 niti tako da jedna od niti u toku svog izvrsavanja prekine
izvrsavanje druge niti.
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */

class NTZ1 implements Runnable {
    NTZ1(NTZ1 n1, String ime) {
        n = n1;
        nit = new Thread(this, ime);
        nit.start();
    }
    public void run() {
        try {
            nit.sleep(5000);
        } catch (InterruptedException ie) {
        }
        System.out.println("Zavrsena je " + nit.getName() + " nit!");
    }
    void prekiniDruguNit() {
        n.nit.interrupt();
    }
    public static void main(String args[]) {
        NTZ1 n1 = new NTZ1(null, "prva");
        NTZ1 n2 = new NTZ1(n1, "druga");
        n2.prekiniDruguNit();
    }
    Thread nit;
    NTZ1 n;
}

```

```
/*
Rezultat:
Završena je prva nit
Završena je druga nit
*/
```

```
/*
Primer NTZ2: Napisati 2 niti tako da jedna od niti u toku svog izvrsavanja uspava
drugu nit.
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */

class NTZ2 extends Thread {
    NTZ2(NTZ2 n1, String ime) {
        super(ime);
        n = n1;
        start();
    }
    public void run() {
        while (signal) {
            if (getName().equals("prva") == true) {
                System.out.println("Prva nit se izvrsava!!!!");
            }
            if (getName().equals("druga") == true) {
                System.out.println("Druga nit se izvrsava!!!!");
            }
        }
    }
    void promeni() {
        signal = false;
    }
    void uspavaj() throws InterruptedException {
        System.out.println("Uspavana je prva nit");
        n.sleep(10);
        System.out.println("Probudjena je prva nit");
    }
    public static void main(String args[]) throws InterruptedException {
        NTZ2 n1 = new NTZ2(null, "prva");
        NTZ2 n2 = new NTZ2(n1, "druga");
        n2.uspavaj();
        n1.promeni();
        n2.promeni();
    }
    boolean signal = true;
    NTZ2 n;
}

/*
Rezultat:
Upavana je prva nit
Prva nit se izvrsava
Probudjena je prva nit
*/
```

Питања:

1. Да ли у наведеном примеру прва нит у току свог извршења успављује другу нит (погледати резултат)?
2. Зашто прва нит у наведеном примеру није успавала другу нит?
3. Да ли нит, која је креирана преко главног програма, наставља да се извршава и након извршења главног програма?

```

/*
Primer NTZ21: Napisati 2 niti tako da jedna od niti u toku svog izvrsavanja uspava
drugu nit.
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */

class NTZ21 extends Thread {
    NTZ21(NTZ21 n1, String ime) {
        super(ime);
        n = n1;
        start();
    }
    public void run() {
        try {
            while (signal) {
                System.out.println("Izvrsava se " + getName() + " nit");

                if (uspavan == true) {
                    System.out.println("Uspavana je " + getName() + " nit");
                    sleep(1); // 1 msec
                    System.out.println("Probudjena je " + getName() + " nit");
                    uspavan = false;
                }
            }
        } catch (InterruptedException ie) {
        }
    }
    void promeni() {
        signal = false;
    }
    void uspavaj() throws InterruptedException {
        n.uspavan = true;
    }
    public static void main(String args[]) throws InterruptedException {
        NTZ21 n1 = new NTZ21(null, "prva");
        NTZ21 n2 = new NTZ21(n1, "druga");
        n2.uspavaj();
        Thread.sleep(2); // 2 msec
        n1.promeni();
        n2.promeni();
    }
    boolean signal = true;
    boolean uspavan = false;
    NTZ21 n;
}
/*
Rezultat:
Izvrsava se prva nit
Uspavana je prva nit
Izvrsava se druga nit
Izvrsava se druga nit
...
Probudjena je prva nit
*/

```

Задатак NT33: Написати програм који ће да реализације сценарио прекида нити логичком контролом када нит није блокирана.

Задатак NT34: Написати програм који ће да реализације сценарио прекида нити interrupt методом када нит није блокирана.

Задатак NT35: Написати програм који ће да реализације сценарио прекида нити interrupt методом када је нит блокирана.

2.6 Коришћење метода isAlive() и join()

Уколико се жели утврдити, да ли се нит још извршава, користи се метода `isAlive()` класе `Thread`. Наведена метода враћа `true`, уколико се нит још извршава, односно `false`, уколико је нит извршена.

Уколико се жели сачекати са извршењем неког дела програма док се нека од нити не изврши користи се метода `join()` класе `Thread`.

```
/*
Primer NT7: Onemoguciti da se glavna nit izvrsi pre nove
niti koriscenjem metode join().
*/
class NT7 extends Thread {
    NT7(String nn) {
        super(nn);
        start();
    }
    public void run() {
        try {
            sleep(2000);
        } catch (InterruptedException e) {
        }
        System.out.println("Zavrsena nova nit!");
    }
    public static void main(String args[]) throws InterruptedException {
        NT7 nn1 = new NT7("Nova nit");
        nn1.join();
        System.out.println("Zavrsena glavna nit!");
    }
}
/*
Rezultat:
Zavrsena nova nit!
Zavrsena glavna nit!
*/
```

Задаци за вежбање:

```
/*
Primer NTZ71: Omoguciti da se dve nove niti izvrse sekvensijalno.
Pre nego sto pocne da se izvrsava druga nit treba da se izvrsi prva nit.
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */
class NTZ71 extends Thread {
    NTZ71(String nn) {
        super(nn);
        start();
    }
    public void run() {
        try {
            sleep(2000);
        } catch (InterruptedException e) {
        }
        System.out.println("Zavrsena nit " + getName());
    }
    public static void main(String args[]) throws InterruptedException {
        NTZ71 n1 = new NTZ71("Nit1");
        n1.join();
        NTZ71 n2 = new NTZ71("Nit2");
    }
}
/*
Rezultat:
Zavrsena nit Nit1
Zavrsena nit Nit2
*/
```

```
/*
Primer NTZ72: Pokrenuti dve nove niti. Iz glavnog programa blokirati jednu nit
dok se ne izvrsi druga nit koriscenjem metode join().
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */
class NTZ72 extends Thread {
    NTZ72(String naziv) {
```

```

        super(naziv);
        start();
    }
    public void run() {
        int i = 1;
        try {
            while (i < 6) // Sta bi se desilo kada bi umesto i<6 stavili i<100
                // a metodu sleep postavili na 1 msec?
            {
                if (uspavan == true) {
                    System.out.println("Uspavana je nit " + getName());
                    sleep(1000);
                } else {
                    System.out.println(i + "-ti prolaz niti" + getName());
                    i++;
                }
            }
        } catch (InterruptedException ie) {
        }
        System.out.println("Zavrsena nit " + getName());
    }

    public static void main(String args[]) throws InterruptedException {
        NTZ72 n1 = new NTZ72("Nit1");
        NTZ72 n2 = new NTZ72("Nit2");
        n1.uspavan = true;
        n2.join();
        n1.uspavan = false;
    }
    boolean uspavan = false;
}
/*
Rezultat:
Uspavana je nit Nit1
1-ti prolaz nitiNit2
2-ti prolaz nitiNit2
3-ti prolaz nitiNit2
4-ti prolaz nitiNit2
5-ti prolaz nitiNit2
Zavrsena nit Nit2

1-ti prolaz nitiNit1
2-ti prolaz nitiNit1
3-ti prolaz nitiNit1
4-ti prolaz nitiNit1
5-ti prolaz nitiNit1
Zavrsena nit Nit1
*/

```

```

/*
Zadatak NTZ73: Kreirati novu nit i iz glavnog programa
pratiti kada ce se ona zavrsiti.
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */
class NTZ73 extends Thread {
    NTZ73(String naziv) {
        super(naziv);
        start();
    }
    public void run() {
        try {
            sleep(1);
        } catch (InterruptedException ie) {
        }
    }
    public static void main(String args[]) throws InterruptedException {
        NTZ73 n1 = new NTZ73("Nit1");
        while (true) {
            if (n1.isAlive() == true) {
                System.out.println("Nit1 jos nije izvrsena");
            } else {
                System.out.println("Nit1 je izvrsena");
                break;
            }
        }
    }
}

```

```

        }
    }
/*
Rezultat:
Nit1 jos nije izvrsena
Nit1 jos nije izvrsena
...
Nit1 je izvrsena
*/

```

Задатак НТ374: Омогућити да се две нове нити изврше секвенцијално без коришћења `join()` методе.

2.7 Приоритет извршавања нити

У Јави свака нит има свој приоритет. Подразумевано нит наслеђује приоритет родитељске нити. Приоритет нити може да се промени са методом `setPriority()`. Приоритет може да узме једну од вредности из опсега од 1 (`MIN_PRIORITY`) до 10 (`MAX_PRIORITY`). Подразумевани приоритет сваке нити, осим ако се другачије не зада је 5 (`NORM_PRIORITY`).

Нит неког нивоа приоритета се извршава док:

- се не позове метода `yield()` која прекида извршење текуће нити, допуштајући да друга нит, истог или вишег приоритета почне да се извршава.
- не пређе у стање “блокиран” или “сршен”.
- нит вишег приоритета не постане извршна (зато што се “пробудила” или је у/т операција комплетирана или је неко позвао `notify()` методу).

Уколико постоји више нити истог приоритета, поставља се питање о редоследу извршења нити. Јава не гарантује да ће све нити, истог приоритета, бити једнако третиране (да ће све добити исто процесорско време – у пракси они добијају приближно исто време, али никада исто). Извесно је да ће свака нит, истог приоритета добити шансу да се паралелно извршава. Нит вишег приоритета добија више процесорског времена од нити нижег приоритета. То значи да ће нит вишег приоритета брже да се изврши од нити нижег приоритета.

Оно што представља реалан проблем при извршењу вишенинтног Јава програма (у смислу транспарентности његовог извршења на различитим оперативним системима), се односи на однос између броја нивоа приоритета нити код Јаве и броја нивоа приоритета нити конкретног оперативног система на коме ће бити извршен Јава програм. На пример оперативни систем NT 4.0 има 7 нивоа приоритета док Јава има 10 нивоа приоритета. Поставља се питање како ће се нивои приоритета из Јаве пресликati у нивое приоритета NT 4.0 оперативног система. Проблем ће се јавити ако се различити нивои приоритета код Јаве пресликају у исти ниво приоритета код NT 4.0 оперативног система.

```

/*
Primer NT8: Napisati program koji ce da procentualno pokaze zauzetost procesora
od strane vise niti, u zavisnosti od prioriteta niti.
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

import java.text.*;
class NT8 extends Thread {
    NT8(String nn, int Prioritet) {
        super(nn);
        setPriority(Prioritet);
        start();
    }
}

```

```

public void run() {
    while (signal) {
        brojac++;
    }
}
void promeni() {
    signal = false;
}
public void prekini() {
    signal = false;
}
public static void main(String args[]) throws InterruptedException {
    long SumaKoraka;
    NumberFormat nf = NumberFormat.getNumberInstance();

    NT8 nn1 = new NT8("Nit1", 5); // Visi prioritet
    NT8 nn2 = new NT8("Nit2", 4); // Nizi prioritet
    Thread.sleep(10000); // Sto je veci broj sekundi (kada je isti prioritet-5)
    // odnos vremena tezi ka 50:50 inace ce da se povecava odnos vremena u
    // korist jedne od niti
    nn1.promeni();
    nn2.promeni();
    nn1.join();
    nn2.join();

    SumaKoraka = nn1.brojac + nn2.brojac;
    double p1 = (double) nn1.brojac / SumaKoraka * 100.00;
    double p2 = (double) nn2.brojac / SumaKoraka * 100.00;

    System.out.println("Nit 1 - broj koraka:" + nn1.brojac + " Procenat:" + p1);
    System.out.println("Nit 2 - broj koraka:" + nn2.brojac + " Procenat:" + p2);
}
private boolean signal = true;
long brojac = 0;
}
/*
Rezultat:
a) 10 sekundi se izvrsavaju niti
Nit 1 ce zauzeti priblizno 98% procesorskog vremena ako ima prioritet 10
Nit 2 ce zauzeti priblizno 2% procesorskog vremena ako ima prioritet 1
b) 10 sekundi se izvrsavaju niti
Nit 1 ce zauzeti priblizno 50% procesorskog vremena ako ima prioritet 5
Nit 2 ce zauzeti priblizno 50% procesorskog vremena ako ima prioritet 5
c) 10 sekundi se izvrsavaju niti
Nit 1 ce zauzeti priblizno 97% procesorskog vremena ako ima prioritet 5
Nit 2 ce zauzeti priblizno 3% procesorskog vremena ako ima prioritet 4
*/

```

Задатак за вежбање:

Задатак NTZ4: Написати 2 нити са различитим приоритетима извршавања. У току рада измените приоритетете између нити. Показате како је трошено процесорско време од стране нити пре и после промене приоритета.

2.8 Себичне (*selfish*) нити

Нити би требале периодично да позивају `yield()` или `sleep()` методу како би другим нитима пружиле шансу да се изврше. На тај начин нит укида могућност да из неког разлога има монопол над системом. Оне нити које не дају другим нитима могућност да се изврше, називају се “себичне” нити.

2.9 Групе нити

Неки програми могу да садрже велики број нити. Тада је пожељно да се нити категоризују по функционалности, како би се њима лакше управљало. На пример, у случају интернет читача, који “скида” *web* страну са више слика, читач извршава више нити, за сваку слику по једна. Уколико корисник притисне дугме “*stop*”, тада ће се прекинути пуњење текуће слике (прекинуће се извршење текуће нити). Тада је потребно да се и остale нити прекину.

Јава програмски језик, омогућава формирање групе нити (*thread group*).

Група нити се креира на следећи начин:

```
String imeGrupe = "novaGrupa";
ThreadGroup g = new ThreadGroup(imeGrupe);
```

Стринг који треба да идентификује групу мора имати особину једнозначности у односу на имена других група нити.

Нит, која ће се звати "novaNit1" се додаје до групе g на следећи начин:

```
Thread t = new Thread(g,"novaNit1");
```

Уколико се жели проверити да ли је нека од нити још у стању "извршно", користи се метода activeCount() на следећи начин:

```
if (g.activeCount() == 0) {
    // sve niti u grupi niti su zastavljene.
}
```

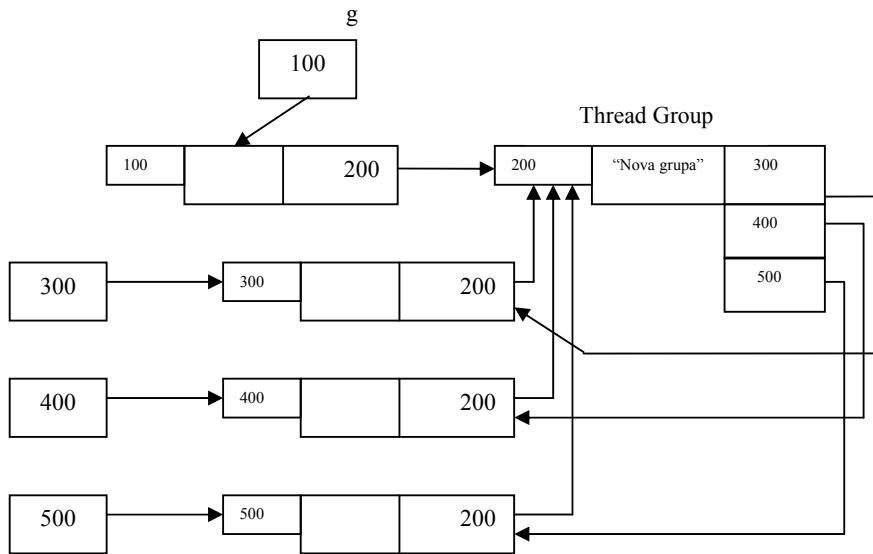
Уколико се желе прекинути све нити у групи нити позива се interrupt() метода над групом нити:

```
g.interrupt();
```

Група нити може да има своју подгрупу нити. Методе activeCount() и interrupt() се односе на текућу групу нити и све њене подгрупе нити. То значи, нпр. да када се прекида извршење једне групе нити, све њене подгрупе нити такође прекидају извршење.

```
/*
Primer NT9: Pokazati kako se kreira i prekida grupa niti.
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */
class NT9 extends Thread {
    NT9(NT9G g, String imeNiti) {
        super(g.tg, imeNiti); // Vrsi se povezivanje izmedju grupe niti i konkretnie niti.
        start();
    }
    public void run() {
        while (!interrupted() && true) {
        }
        System.out.println("Prekinuta nit");
    }
    public static void main(String args[]) throws Exception {
        String imeGrupe = "novaGrupa";
        NT9G g = new NT9G(imeGrupe);
        NT9 t1 = new NT9(g, "novaNit1");
        NT9 t2 = new NT9(g, "novaNit2");
        NT9 t3 = new NT9(g, "novaNit3");
        g.prekini();
    }
}
class NT9G extends Thread {
    NT9G(String imegrupa) {
        tg = new ThreadGroup(imegrupa);
        start();
    }
    public void run() {
        while (true) {
            if (tg.activeCount() == 0) {
                System.out.println("Sve niti u grupi su prekinute! " + tg.activeCount());
                break;
            }
        }
    }
    void prekini() {
        tg.interrupt();
    }
    ThreadGroup tg;
}
```

Изглед оперативне меморије задатка NT9:



Задаци за вежбање:

```
/*
Primer NTZ5: Napisati grupu niti koja prati stanje nekog objekta.
Kada objekat dobije zadatu vrednost prekinuti izvršenje grupe niti.
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */

class Zalihe {
    int kolicina = 10;
    void smanji() {
        kolicina--;
    }
    int vratiKolicinu() {
        return kolicina;
    }
}

class NTZ5 extends Thread {
    NTZ5(NTZ5G g, String imeNiti, Zalihe z1) {
        super(g.tg, imeNiti);
        z = z1;
        start();
    }
    public void run() {
        smanji();
    }

    void smanji()// Kada se ubaci synchronized tada ne moze da se desi da se izvrsi nit
    // koja je usla u metodu posle neke druge niti.
    {
        while (!interrupted() && true) {
            z.smanji();
            System.out.println("Kolicina na zalihama: " + z.vratiKolicinu() + " nit " +
getName());
            if (z.vratiKolicinu() <= 0) {
                getThreadGroup().interrupt();
            }
        }
        System.out.println("Prekinuta nit " + getName());
    }
}
```

```

public static void main(String args[]) throws Exception {
    String imeGrupe = "novaGrupa";
    Zalihe z = new Zalihe();
    NTZ5G g = new NTZ5G(imeGrupe);
    NTZ5 t1 = new NTZ5(g, "novaNit1", z);
    NTZ5 t2 = new NTZ5(g, "novaNit2", z);
    NTZ5 t3 = new NTZ5(g, "novaNit3", z);

}
Zalihe z;
}

class NTZ5G extends Thread {
    NTZ5G(String imegrupe) {
        tg = new ThreadGroup(imegrupe);
        start();
    }
    public void run() {
        while (true) {
            if (tg.activeCount() == 0) {
                System.out.println("Sve niti u grupi su prekinute!");
                break;
            }
        }
    }
    ThreadGroup tg;
}
/*
Rezultat: (je promenljiv jer se ne moze predvideti koja nit ce da spusti kolicinu zaliha na 0)
Kolicina na zalihamama: 9 nit novaNit1
Kolicina na zalihamama: 8 nit novaNit1
Kolicina na zalihamama: 7 nit novaNit1
Kolicina na zalihamama: 6 nit novaNit1
Kolicina na zalihamama: 5 nit novaNit1
Kolicina na zalihamama: 4 nit novaNit1
Kolicina na zalihamama: 3 nit novaNit1
Kolicina na zalihamama: 2 nit novaNit1
Kolicina na zalihamama: 0 nit novaNit2
Prekinuta nit novaNit2
Kolicina na zalihamama: 1 nit novaNit3 // Nit 3 je usla pre niti 2 u metodu a posle nje se izvrsila.
Prekinuta nit novaNit3
Prekinuta nit novaNit1
Sve niti u grupi su prekinute!
*/

```

2.10 Синхронизација

Уколико се јави потреба да две или више нити деле заједнички ресурс, при чему нити не могу истовремено да користе заједнички ресурс, мора се обезбедити механизам које ће омогућити искључиви (ексклузивни) приступ једне нити до заједничког ресурса. Тек након завршетка обраде заједничког ресурса од стране једне нити, могуће је да друга нит, такође искључиво, приступи до њега.

Монитор обезбеђује механизам за искључиви приступ нити до заједничког ресурса. Када нека нит X уђе у монитор, ниједна друга нит не може у њега ући, све док нит X не изађе из њега. Поступак којим монитор обезбеђује наведени механизам назива се синхронизација.

Потреба истовременог приступа до дељеног објекта доводи до тзв. *race condition*¹⁶ ситуација.

2.10.1 Комуникација нити без синхронизације

Онемогућавање истовременог приступа нити до дељених објеката се назива синхронизација приступа. Уколико нема синхронизације приступа може се јавити следећа ситуација:

¹⁶ Када на тркама 2 такмичара, скоро у исто време пролазе кроз циљ, јавља се проблем прецизног одређивања ко је први прошао кроз циљ.

Уколико постоји банка са 10 рачуна (за сваки рачун се прави посебна нит), између којих се врши пренос новца на случајно изабран начин, може се десити да износ истог рачуна истовремено повећава 2 или више нити.

Нпр. `racun[5]` се истовремено повећава помоћу две нити.

Стање рачуна пре промене: `racun[5] = 500;`

Прва нит: `racun[5] +=100;`

Друга нит: `racun[5] +=50;`

Проблем се јавља на нивоу атомских операција наредби:

Прва нит: пуни се меморијски регистар `MR1` са износом од `racun[5]` преко наредбе `load()`.

`MR1 = 500`

`racun[5] = 500`

Након тога се повећава износ регистра за 100 преко наредбе `add()`.

`MR1 = 600`

`racun[5] = 500`

Уколико тада почне да се извршава друга нит, тада се пуни меморијски регистар `MR2` са износом од `racun[5]` са наредбом `load()`.

`MR2 = 500`

`racun[5] = 500`

Након тога се повећава износ регистра за 50 преко наредбе `add()`.

`MR2 = 550`

`racun[5] = 500`

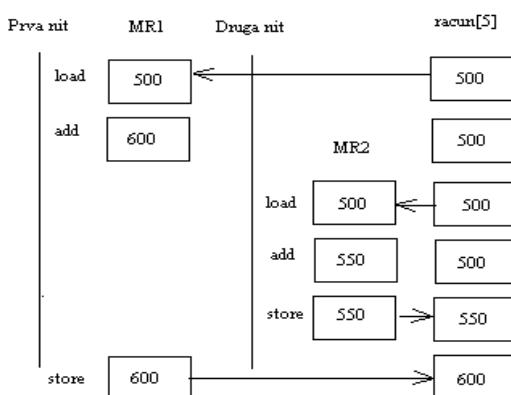
Ако тада `racun[5]` добије вредност регистра `MR2` са наредбом `store()`.

`racun[5] = 550`

На крају ће `racun[5]` добити вредност регистра `MR1` са наредбом `store()`.

`racun[5] = 600`

Добијени износ је погрешан. Валидна вредност коју би `racun[5]` требао да има је 650.



2.10.2 Закључавање објеката

Када нит позове методу објекта који треба да се синхронизује, објекат постаје закључан. Прецизније речено при позиву синхронизоване методе: "Нит налази једини кључ од објекта испред врата, убацује кључ у браву од објекта, откључава га, улази у објекат и браву са друге стране врата закључава. Брава остаје закључана све док нит не изађе из објекта, односно док нит не извади кључ из браве и стави га испред врата". Нити које чекају на приступ синхронизованим методама дешеног објекта, могу приступити до несинхронизованих метода дешеног објекта.

2.10.3 Коришћење синхронизованих метода

У Јави сваки објекат има свој монитор. Да би се монитор покренуо потребно је да се објекту приступи преко једне од његових синхронизованих метода. Испред метода које треба синхронизовати ставља се кључна реч **synchronized**. Треба нагласити да извршавање једне од синхронизованих метода неког објекта онемогућава приступ до било које друге синхронизоване методе истог објекта. Тако да након завршетка извршења синхронизоване методе неког објекта могуће је позвати неку другу синхронизовану методу истог објекта. За несинхронизоване методе не важи наведено ограничење.

```
/*
 * Primer NT10S1: Omoguciti sinhronizaciju objekta, kome pristupa vise niti.*/
*/
* @author Sinisa Vlajic
* SILAB - Labartorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

class Nit extends Thread {
    Nit(String Naziv, int BrojNiti, Zalihe z) {
        this.Naziv = Naziv;
        this.BrojNiti = BrojNiti;
        this.z = z;
        start();
    }
    public void run() {
        int PreostalaKolicina = z.Uzmi();
        System.out.println("Nit:" + BrojNiti + " Preostala kolicina:" + PreostalaKolicina);
    }
    int BrojNiti;
    String Naziv;
    Zalihe z;
}

class Zalihe {
    private int Kolicina;
    Zalihe() {
        Kolicina = 10;
    }
    int Uzmi() // Ukoliko se stavi synchronized int Uzmi() nece biti problema u rezultatu.
    {
        Kolicina--;
        try {
            Thread.sleep(1000); // 1 sekunda pauze
        } catch (InterruptedException e) {
            System.out.println("Prekid niti");
        }
        return Kolicina;
    }
}

class NT10S1 {
    public static void main(String args[]) {
        Zalihe z = new Zalihe();
        Nit nn1 = new Nit("Nit1", 1, z);
        Nit nn2 = new Nit("Nit2", 2, z);
        Nit nn3 = new Nit("Nit3", 3, z);
    }
}
/*
Rezultat:
Nit 1: Preostala kolicina:7
Nit 2: Preostala kolicina:7
Nit 3: Preostala kolicina:7
*/
```

Проблем је у томе што се не види ефекат једне од нити када се врати резултат, већ се види ефекат извршења свих нити. Наведени резултат показује да није извршена

синхронизација објекта¹⁷ з преко методе `Uzmi()`. Да би се постигао наведени ефекат потребно је ставити **synchronized** int `Uzmi()` у класи `Zalihe`, уместо int `Uzmi()`:

```
/* primer: Sinhronizacija.java */
class Zalihe{
    private int Kolicina;
    Zalihe() {
        Kolicina = 10;
    }
    synchronized int Uzmi() {
        ...
    }
}

tada će rezultat program biti sledeći:
/*
Rezultat:
Nit 1: Preostala kolicina:9
Nit 2: Preostala kolicina:8
Nit 3: Preostala kolicina:7
*/
```

Задатак NTZ6: Коришћењем синхронизације онемогућити да објекту класе `Student` приступе истовремено 2 нити и промене његове податке.

2.10.4 Коришћење синхронизованих објеката

У случају да преко метода не може да се синхронизује приступ објекту, могуће је експлицитно синхронизовати сам објекат. Општи облик експлицитне синхронизације објекта је:

```
synchronized (objekat) {
    blok naredbi koje će biti sinhronizovane
}
```

Уколико се жели постићи експлицитна синхронизација објекта, потребно је у предходном примеру да се метода `run()` класе `Nit` замени са:

```
/* primer: NT10S2.java */
class Nit extends Thread{
    ...
    public void run(){
        int PreostalaKolicina;
        synchronized (z){
            PreostalaKolicina = z.Uzmi();
        }
        System.out.println("Nit:" + BrojNiti + " Preostala kolicina:" + PreostalaKolicina);
    }
    ...
}

/*
Rezultat:
Nit 1: Preostala kolicina:9
Nit 2: Preostala kolicina:8
Nit 3: Preostala kolicina:7
*/
```

Када се синхронизација објекта врши преко методе, тада се пре извршења синхронизоване методе закључава објекат. У том статусу објекат остаје све док се не изврши синхронизована метода.

¹⁷ Када се каже да се објекат синхронизује, то значи да се њему не може приступити истовремено преко две или више нити.

Када се синхронизација објекта врши експлицитно, тада се пре извршења блока наредби које треба синхронизовати закључава објекат. У том статусу објекат остаје све док се не изврши наведени блок наредби.

2.11 Комуницирање између нити

Комуникација између нити у Јави се постиже помоћу метода `wait()`, `notify()` и `notifyAll()` на следећи начин:

Када нит која уђе у синхронизовану методу, због неког услова не може да изврши комплетну методу, она ће или изаћи из методе необављена посла или ће остати у методи (у блокираном статусу) чекајући да је нека друга нит опет покрене. У блокирани статус метода прелази помоћу методе `wait()`. Тада нит напушта монитор у коме се налази и прелази у листу чекања (wait list). У блокираном статусу нит остаје све док нека друга нит не позове методу `notify()` или `notifyAll()`. Тада се нит опет враћа у монитор напуштајући листу чекања.

Метода `notify()` позива једну од нити из листе чекања, док метода `notifyAll()` позива све нити из листе чекања. Када више нити истовремено из листе чекања покуша да се врати у монитор, само ће једна од њих ући у монитор, док ће се остале вратити у листу чекања. Нити вишег приоритета прве ће се вратити у монитор из листе чекања. Када су нити истог приоритета, не постоји правило које може да прецизира која нит има првенство у односу на друге нити истог приоритета.

Методе `wait()`, `notify()` и `notifyAll()` могу се позвати једино из синхронизованих метода.

У следећем примеру две нити ће позивати две различите синхронизоване методе `Uzmi()` и `Stavi()`. На редослед позивања ових метода у овом случају се не може утицати. Тако ће једна од нити прва позвати неколико пута заредом своју методу, па ће тек онда друга нит да позове своју методу неколико пута заредом.

```
/*
Primer NT10S3: Napisati program koji ce izvrsavati dve niti koje ce pozivati
dve razlicite sinhronizaovane metode.Jedna nit moze da pozove svoju
metodu nekoliko puta zaredom.
*/
import java.io.*;
class Proizvodjac extends Thread {
    Proizvodjac(String Naziv, Zalihe z) {
        this.Naziv = Naziv;
        this.z = z;
        start();
    }
    public void run() {
        for (int i = 0; i < 4; i++) {
            z.Stavi(i);
        }
    }
    String Naziv;
    Zalihe z;
}

class Potrosac extends Thread {
    Potrosac(String Naziv, Zalihe z) {
        this.Naziv = Naziv;
        this.z = z;
        start();
    }
    public void run() {
        for (int i = 0; i < 4; i++) {
            z.Uzmi(i);
        }
    }
}
```

```

        String Naziv;
        Zalihe z;
    }
    class Zalihe {
        boolean signal = false;
        OutputStream dat;
        Zalihe() {
            try {
                dat = new FileOutputStream("izlaz.txt");
            } catch (Exception e) {
                System.out.println("Izuzetak kod otv. dat!");
            }
        }
        synchronized void Uzmi(int i) {
            ZapamtiUDat("Uzmi " + i);
        }
        synchronized void Stavi(int i) {
            ZapamtiUDat("Stavi " + i);
        }
        void ZapamtiUDat(String poruka) {
            try {
                poruka = poruka + "\n";
                dat.write(poruka.getBytes());
            } catch (IOException io) {
                System.out.println("UI izuzetak:" + io);
            }
        }
    }
    class NT10S3 {
        public static void main(String args[]) {
            Zalihe z = new Zalihe();
            Proizvodjac nn1 = new Proizvodjac("Nit1", z);
            Potrosac nn2 = new Potrosac("Nit2", z);
            try {
                nn1.join();
                nn2.join();
            } catch (InterruptedException e) {
                System.out.println("Prekid glavne niti");
            }
            System.out.println("Zavrsetak programa!");
        }
    }

/*
Rezultat: u datoteci izlaz.txt
Stavi 0
Stavi 1
Stavi 2
Stavi 3
Uzmi 0
Uzmi 1
Uzmi 2
Uzmi 3
*/

```

Уколико се жели постићи ефекат да 2 нити наизменично позивају своје методе треба урадити следеће:

```

/*
Primer NT10S31: Napisati program kod koga ce 2 niti
naizmenicno pozivati svoje metode.
*/

import java.io.*;
class Proizvodjac extends Thread {
    Proizvodjac(String Naziv, Zalihe z) {
        this.Naziv = Naziv;
        this.z = z;
        start();
    }
    public void run() {
        for (int i = 0; i < 4; i++) {
            z.Stavi(i);
        }
        z.Deblokiraj();
    }
    String Naziv;
    Zalihe z;
}

```

```

class Potrosac extends Thread {
    Potrosac(String Naziv, Zalihe z) {
        this.Naziv = Naziv;
        this.z = z;
        start();
    }
    public void run() {
        for (int i = 0; i < 4; i++) {
            z.Uzmi(i);
        }
        z.Deblokiraj();
    }
    String Naziv;
    Zalihe z;
}

class Zalihe {
    synchronized void Uzmi(int i) {
        try {
            notify();
            System.out.println("Uzmi " + i);
            wait();
        } catch (Exception e) {
        }
    }
    synchronized void Stavi(int i) {
        try {
            notify();
            System.out.println("Stavi " + i);
            wait();
        } catch (Exception e) {
        }
    }
    synchronized void Deblokiraj() {
        notifyAll();
    }
}

class NT10S31 {
    public static void main(String args[]) {
        Zalihe z = new Zalihe();
        Proizvodjac nn1 = new Proizvodjac("Nit1", z);
        Potrosac nn2 = new Potrosac("Nit2", z);
        try {
            nn1.join();
            nn2.join();
        } catch (InterruptedException e) {
            System.out.println("Prekid glavne niti");
        }
        System.out.println("Zavrsetak programa!");
    }
}
/*
Rezultat:
Stavi 0
Uzmi 0
Stavi 1
Uzmi 1
Stavi 2
Uzmi 2
Stavi 3
Uzmi 3
*/

```

У следећем примеру NT10S41 се онемогућава произвођачу и потрошачу да преко метода `Uzmi()` и `Stavi()` подигну стање залиха више од 10 и мање од 0.

```

import java.io.*;
class Proizvodjac extends Thread {
    Proizvodjac(String Naziv, Zalihe z) {
        this.Naziv = Naziv;
        this.z = z;
        start();
    }
    public void run() {
        int i = 0;
        while (true) {
            z.Stavi(i);
        }
    }
}

```

```

        i++;
    }
}
String Naziv;
Zalihe z;
}
class Zalihe{
    synchronized void Uzmi(int i){
    try {
        notify();
        System.out.println("Uzmi " + i);
        wait();
    }catch(Exception e){}
    }
    synchronized void Stavi(int i){
    try {
        notify();
        System.out.println("Stavi " + i);
        wait();
    }catch(Exception e){}
    }
    synchronized void Deblokiraj() {
        notifyAll();
    }
}
class Potrosac extends Thread {
    Potrosac(String Naziv, Zalihe z) {
        this.Naziv = Naziv;
        this.z = z;
        start();
    }
    public void run() {
        int i = 0;
        while (true) {
            z.Uzmi(i);
            i++;
        }
    }
    String Naziv;
    Zalihe z;
}

class NT10S41
{ public static void main(String args[]){
    Zalihe z = new Zalihe();
    Proizvodjac nn1 = new Proizvodjac("Nit1", z);
    Potrosac nn2 = new Potrosac("Nit2", z);
    try {
        nn1.join();
        nn2.join();
    } catch (InterruptedException e) {
        System.out.println("Prekid glavne niti");
    }
    System.out.println("Zavrsetak programa!");
}
}

```

Питање:

1. Када је нит блокирана у синхронизованој методи неког објекта, да ли нека друга нит може приступити том објекту преко друге или исте синхронизоване методе тог објекта?

Програм **NT10S5.java** бави се проблемом подизања и спуштања стања залиха робе у конкурентном окружењу једног произвођача и више купаца. У наведеном задатку, слично примеру `Sinhronizacija3.java`, синхронизоване методе се наизменично позивају.

```

/*
Primer NT10S5: Napraviti 3 niti, od kojih ce jedna da podize stanje zaliha (metoda Stavi()), za 1 komad, dok ce preostale dve niti da skidaju stanje zaliha (metoda Uzmi()). Jedna nit ce da smanjuje stanje za 2 komada, dok ce druga nit da smanjuje stanje za 3 komada.Omoguciti da se sinhronizovane metode Stavi() i Uzmi() naizmenicno izvrsavaju.
*/
/**
 * @author Sinisa Vlajic
 * SILAB - Labartorija za Softversko Inzenjerstvo

```

```

* FON - Beograd
*http://silab.fon.bg.ac.yu
*/
import java.io.*;
class Proizvodjac extends Thread {
    Proizvodjac(String Naziv, Zalihe z, int Kol) {
        this.Naziv = Naziv;
        this.z = z;
        KolDaje = Kol;
        start();
    }
    public void run() {
        while (signal) {
            z.Stavi(KolDaje, Naziv);
        }
    }
    void prekini() {
        signal = false;
    }
    String Naziv;
    Zalihe z;
    int KolDaje;
    volatile boolean signal = true;
}
class Potrosac extends Thread {
    Potrosac(String Naziv, Zalihe z, int Kol, int Prioritet) {
        this.Naziv = Naziv;
        this.z = z;
        setPriority(Prioritet);
        KolUzima = Kol;
        start();
    }
    public void run() {
        while (signal) {
            z.Uzmi(KolUzima, Naziv);
        }
    }
    void prekini() {
        signal = false;
    }
    String Naziv;
    Zalihe z;
    int KolUzima;
    volatile boolean signal = true;
}

class Zalihe {
    private int Kolicina;
    boolean signal = false;
    OutputStream dat;
    Zalihe() {
        Kolicina = 0;
        try {
            dat = new FileOutputStream("izlaz.txt");
        } catch (Exception e) {
            System.out.println("Izuzetak kod otv. dat!");
        }
    }
    synchronized void Uzmi(int Kol, String naziv) {
        if (!signal) {
            try {
                ZapamtiUDat("wait-uzmi ulaz: " + naziv);
                wait();
                ZapamtiUDat("wait-uzmi izlaz: " + naziv);
            } catch (InterruptedException e) {
                System.out.println("Izuzetak!");
            }
        }
        signal = false;
        if (Kolicina - Kol >= 0) {
            Kolicina = Kolicina - Kol;
            ZapamtiUDat("Potrosac:" + naziv + "Kolicina:" + Kolicina);
        } else {
            ZapamtiUDat("Potrosac-bezuspesno:" + naziv + "Kolicina:" + Kolicina);
        }
        notifyAll();
    }
    synchronized void Stavi(int Kol, String naziv) {

```

```

        if (signal) {
            ZapamtiUDat("wait-stavi ulaz: " + naziv);
            try {
                wait();
                ZapamtiUDat("wait-stavi izlaz: " + naziv);
            } catch (InterruptedException e) {
                System.out.println("Izuzetak!");
            }
        }
        signal = true;
        Kolicina = Kolicina + Kol;
        ZapamtiUDat("Proizvodjac:" + naziv + "Kolicina:" + Kolicina);
        notifyAll();
    }

    void ZapamtiUDat(String poruka) {
        try {
            poruka = poruka + "\n";
            dat.write(poruka.getBytes());
        } catch (IOException io) {
            System.out.println("UI izuzetak:" + io);
        }
    }
}
class NT10S5 {

    public static void main(String args[]) {
        Zalihe z = new Zalihe();
        Proizvodjac pr1 = new Proizvodjac("NIT-PROIZVODJAC ", z, 1);
        Potrosac pot1 = new Potrosac("NIT-POTROSAC1 ", z, 2, 5);
        Potrosac pot2 = new Potrosac("NIT-POTROSAC2 ", z, 3, 5);

        try {
            Thread.sleep(100);
        } catch (Exception e) {
        }
        pr1.prekini();
        pot1.prekini();
        pot2.prekini();
        try {
            pr1.join();
            pot1.join();
            pot2.join();
        } catch (InterruptedException e) {
            System.out.println("Prekid glavne niti");
        }
        System.out.println("Izadjite iz programa!");
    }
}
// Rezultat:Pogledati u datoteci izlaz.txt

```

Програм NT10S6.java се бави проблемом издавања карти за аутобус у конкурентном окружењу више купаца.

```

/*
Programski zahtev: Napraviti program koji ce da prati izdavanje karata za autobus.
Autobus moze da ima najvise 52 mesta. Ukupno ima 120 potencijalnih kupaca karata.
Za svakog potencijalnog kupca pokrenuti nit koja ce biti aktivna sve dok se ili
ne zavrssi program ili dok kupac ne kupi kartu. Takodje pokrenuti 20 niti za
proizvoljnih 20 kupaca koji ce vratiti kartu. Niti koje prate kupce koji vracaju
karte ce biti aktivne sve dok se ili ne zavrssi program ili dok kupac ne vrati
kartu. Na kraju prikazati u datoteci izlaz.txt zauzeta mesta u autobusu
i ime kupca koji je kupio zauzeo to mesto.
*/
import java.io.*;
import java.util.Random;

class KupovinaKarte extends Thread {

    KupovinaKarte(String ImePutnika, Autobus aut) {
        this.ImePutnika = ImePutnika;
        setPriority(5);
        this.aut = aut;
        start();
    }

    public void run() {

```

```

        while (!aut.Uzmi(ImePutnika)) {
            try {
                Thread.sleep(45);
            } catch (Exception e) {
            }
        }
    String ImePutnika;
    Autobus aut;
}

class VracanjeKarte extends Thread {

    VracanjeKarte(String ImePutnika, Autobus aut) {
        this.ImePutnika = ImePutnika;
        this.aut = aut;
        setPriority(5);
        start();
    }

    public void run() {
        while (!aut.Vrati(ImePutnika)) {
            try {
                Thread.sleep(15);
            } catch (Exception e) {
            }
        }
    String ImePutnika;
    Autobus aut;
}
}

class Sedista {

    int Mesto;
    boolean Zauzeto = false;
    String ImePutnika;
}
}

class Autobus {

    Sedista sed[];
    OutputStream dat;
    boolean signal = true;

    Autobus() {
        sed = new Sedista[52];
        for (int i = 0; i < 52; i++) {
            sed[i] = new Sedista();
            sed[i].Mesto = i;
            sed[i].ImePutnika = "nema";
        }
        try {
            dat = new FileOutputStream("izlaz.txt");
        } catch (Exception e) {
            System.out.println("Izuzetak kod otv. dat!");
        }
    }

    synchronized boolean Uzmi(String Ime) {
        if (!signal) // Kada se iz main() pozove prekini, da zaustavi nit.
        {
            return true;
        }

        for (int i = 0; i < 52; i++) {
            if (sed[i].Zauzeto == false) {
                sed[i].Zauzeto = true;
                sed[i].ImePutnika = Ime;
                ZapamtiUDat("Putnik " + Ime + " je uzeo kartu.");
                return true;
            }
        }
        ZapamtiUDat("Putnik " + Ime + " NIJE uzeo kartu.");
        return false;
    }

    synchronized boolean Vrati(String Ime) {
        if (!signal) // Kada se iz main() pozove prekini, da zaustavi nit.
        {

```

```

    {
        return true;
    }
    for (int i = 0; i < 52; i++) {
        if (sed[i].ImePutnika.equals(Ime)) {
            sed[i].Zauzeto = false;
            sed[i].ImePutnika = "nema";
            ZapamtiUDat("Putnik " + Ime + "je vratio kartu.");
            return true;
        }
    }
    ZapamtiUDat("Putnik " + Ime + "ima nameru da vrati kartu ali je nije jos kupio.");
    return false;
}

void Prikazi() {
    ZapamtiUDat("Izvestaj o zauzetim mestima:");
    for (int i = 0; i < 52; i++) {
        if (sed[i].Zauzeto) {
            ZapamtiUDat("Putnik " + sed[i].ImePutnika + "ima sediste " + sed[i].Mesto);
        }
    }
}

void ZapamtiUDat(String poruka) {
    try {
        poruka = poruka + "\n";
        dat.write(poruka.getBytes());
    } catch (IOException io) {
        System.out.println("UI izuzetak:" + io);
    }
}

void Prekini() {
    signal = false;
}
}

class NT10S6 {
    public static void main(String args[]) {
        Autobus aut = new Autobus();
        KupovinaKarte kk[] = new KupovinaKarte[120];
        VracanjeKarte vk[] = new VracanjeKarte[20];
        for (int i = 0; i < 120; i++) {
            kk[i] = new KupovinaKarte("Kupac" + i, aut);
        }

        Random r = new Random(0);
        for (int i = 0; i < 20; i++) {
            int vred = (int) (r.nextGaussian() * 51);
            vk[i] = new VracanjeKarte("Kupac" + Math.abs(vred), aut);
        }

        try {
            Thread.sleep(300);
        } catch (Exception e) {
        }

        aut.Prekini();
        try {
            for (int i = 0; i < 120; i++) {
                kk[i].join();
            }
            for (int i = 0; i < 20; i++) {
                vk[i].join();
            }
        } catch (InterruptedException e) {
            System.out.println("Prekid glavne niti");
        }

        aut.Prikazi();
        System.out.println("Izadjite iz programa!");
    }
}

// Rezultat:Pogledati u datoteci izlaz.txt

```

Задатак NTZ7: Направити програм који ће да прати рад 3 нити, тако да свака од нити периодично добије шансу да се изврши. Нити су истог приоритета. У раду користити методе `wait()` и `notify()`.

2.12 Међусобно блокирање нити

Нити могу у конкурентном раду да се међусобно блокирају (*deadlock*). Наведена ситуација настаје у следеће 2 ситуације:

Уколико се једна од нити блокира преко `wait()` методе а друга нит уђе такође преко `wait()` методе у блокаду, не деблокирајући пре тога `notify()` или `notifyAll()` методе прву нит, тада и једна и друга нит бивају блокиране и у том статусу остају док се не прекине програм.

```
// NT10S4.java
import java.io.*;
...
class Zalihe {

    boolean signal = false;
    ...
    synchronized void Uzmi(int i) {
        if (!signal) {
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("Izuzetak!");
            }
        }
        signal = false;
        ZapamtiUDat("Uzmi " + i);
        //notify(); Ukoliko se ne pozove notify() metoda,
        // nikada neće moci da se odblokira nit
        // koja je usla u blokadu preko wait()
        // pri izvršenju sinhronizovane metode Stavi().
    }

    synchronized void Stavi(int i) {
        if (signal) {
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("Izuzetak!");
            }
        }
        signal = true;
        ZapamtiUDat("Stavi " + i);
        //notify(); Ukoliko se ne pozove notify() metoda,
        // nikada neće moci da se odblokira nit
        // koja je usla u blokadu preko wait()
        // pri izvršenju sinhronizovane metode Uzmi().
    }
    ...
}
class NT10S4 {
    ...
}
```

Прва нит позива синхронизовану методу објекта А а друга нит позива синхронизовану методу објекта В. Код извршавања, синхронизована метода објекта А покушава да позове једну од синхронизованих метода објекта В. Такође код извршавања, синхронизована метода објекта В покушава да позове једну од синхронизованих метода објекта А. Тада долази до блокаде која ће трајати све док се не прекине програм.

Задатак NTZ71: Направити програм којим ће се заблокирати међусобно нити сходно сценарију b.

```

/*
Primer NT10S7: Napisati program kod koga ce prva nit pozivati sinhronizovanu metodu objekta A, dok ce druga nit da poziva sinhronizovanu metodu objekta B. Kod izvrsavanja, sinhronizovana metoda objekta A treba da pokusa da pozove jednu od sinhronizovanih metoda objekta B. Takođe kod izvrsavanja, sinhronizovana metoda objekta B treba da pokusava da pozove jednu od sinhronizovanih metoda objekta A.
*/
import java.io.*;
class NIT1 extends Thread {
    NIT1(A a, B b) {
        this.a = a;
        this.b = b;
        start();
    }
    public void run() {
        a.ma1(b);
    }
    A a;
    B b;
}
class NIT2 extends Thread {
    NIT2(A a, B b) {
        this.a = a;
        this.b = b;
        start();
    }
    public void run() {
        b.mb1(a);
    }
    A a;
    B b;
}
class A {
    synchronized void ma1(B b) {
        System.out.println("Metoda ma1()");
        try {
            Thread.sleep(5000);
        } catch (Exception e) {
        }
        b.mb2();
    }
    synchronized void ma2() {
        System.out.println("Metoda ma2()");
    }
}
class B {
    synchronized void mb1(A a) {
        System.out.println("Metoda mb1()");
        try {
            Thread.sleep(5000);
        } catch (Exception e) {
        }
        a.ma2();
    }
    synchronized void mb2() {
        System.out.println("Metoda mb2()");
    }
}
class NT10S7 {
    public static void main(String args[]) {
        A a = new A();
        B b = new B();
        NIT1 n1 = new NIT1(a, b);
        NIT2 n2 = new NIT2(a, b);
        try {
            n1.join();
            n2.join();
        } catch (InterruptedException e) {
            System.out.println("Prekid glavne niti");
        }
        System.out.println("Izadjite iz programa!");
    }
}
/*
Rezultat:
Metoda ma1()
Metoda mb1()
*/

```

Задатак NTZ8: Напишати програм који ће да доведе до блокаде нити. Покушајте да направите пример који ће се разликовати од предходна 2 примера који се баве проблемом блокаде нити.

2.13 Коришћење цеви (pipes) за комуникацију између нити

Комуникација између нити се обавља тако што једна нит, назvana производијач (*producer*), генерише низ бајтова, док друга нит, назvana потрошач, чита тај низ бајтова. Ако бајтови нису расположиви за читање, нит потрошач се блокира. Ако нит производијач генерише више бајтова него што потрошач може да их прочита, нит производијач се блокира. Класе које се користе у комуникацији између нити су: `PipedInputStream` и `PipedOutputStream` (када се разменују бајтови) као и `PipedReader` и `PipedWriter` (када се разменују unicode знаци).

Главни разлог за коришћење цеви је једноставност њиховог рада. Нит производијач шаље бајтове до цеви не знајући која ће их нит прочитати. Нит потрошач чита бајтове из цеви не знајући која нит их је послала у цев. Преко цеви се може повезати међусобно више нити без бриге о њиховој синхронизацији.

```
/*
Primer NT11: Omoguciti komunikaciju izmedju niti proizvodjaca i niti potrosaca
preko cevi (pipes).
*/
import java.util.*;
import java.io.*;

public class NT11 {
    public static void main(String args[]) throws IOException {
        PipedOutputStream cout1 = new PipedOutputStream();
        PipedInputStream cin1 = new PipedInputStream(cout1);
        PipedOutputStream cout2 = new PipedOutputStream();
        PipedInputStream cin2 = new PipedInputStream();
        Proizvodjac pr = new Proizvodjac(cout1);
        Potrosac pot = new Potrosac(cin1);
        pr.start();
        pot.start();
    }
}

class Proizvodjac extends Thread {
    private DataOutputStream out;
    public Proizvodjac(OutputStream os) {
        out = new DataOutputStream(os);
    }
    public void run() {
        try {
            out.writeDouble(67.45);
            out.writeDouble(12.34);
        } catch (Exception ex) {
        }
    }
}

class Potrosac extends Thread {
    private DataInputStream in;
    public Potrosac(InputStream is) {
        in = new DataInputStream(is);
    }
    public void run() {
        try {
            double r1 = in.readDouble();
            double r2 = in.readDouble();
            System.out.println("Potrosac je primio brojeve: " + r1 + " " + r2);
        } catch (Exception ex) {
        }
    }
}
```

Задатак NTZ9: Осмислите пример комуникације између нити помоћу цеви.

3. ПРОГРАМИРАЊЕ (РАД) У МРЕЖИ

3.1 Адреса рачунара

Сваки рачунар на глобалној мрежи (Интернету) има своју адресу (***Internet address***) која се састоји од 4 троцифрена броја, при чему бројеви могу узимати вредност из опсега од 0 – 255. На пример: 132.163.135.130 представља адресу сервера Националног института за стандарде у Колораду.

Повезивање рачунара у мрежу и пренос података између њих се остварује најчешће помоћу **TCP/IP** (***Transmission Control Protocol/Internet Protocol***) протокола. IP је одговоран да пронађе интернет адресу циљног рачунара и да усмери податке ка њему од извornог рачунара. TCP је задужен за успостављање и раскидање везе између рачунара, као и за одређене контролне функције. Често се за интернет адресу каже да је то **IP** адреса.

Класе које омогућавају рад у мрежи у Јави налазе се у пакету `java.net`.

Уколико се жели показати интернет адреса локалне машине користи се метода `getLocalHost()`. Класа која садржи информације о интернет адреси је `InetAddress`.

```
/* Primer Mrezal: Prikazati Internet adresu tekuce masine. */

/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.net.*;
class Mrezal {
    public static void main(String args[]) throws UnknownHostException {
        InetAddress tekucaAdresa = InetAddress.getLocalHost();
        System.out.println(tekucaAdresa);
    }
}
/*
Rezultat:
gsi/147.91.128.109
*/
```

Поред нумеричке адресе рачунара постоји и симболичка адреса рачунара, која се представља преко скупа симбола (знакова). Пример симболичке адресе је: `gsi.fon.bg.ac.yu`. Једна симболичка адреса може бити везана за више нумеричких адреса рачунара.

Симболичка адреса састоји се из два дела:

матичног имена (***host name***) рачунара и
имена домена (***domain name***) коме припада матични рачунар.

На пример симболичка адреса `gsi.fon.bg.ac.yu` састоји се из матичног имена: `gsi` и имена домена: `fon.bg.ac.yu`.

Сервис за именовање домена (***Domain Naming Service - DNS***), повезује симболичке адресе са интернет адресама. *DNS* сервиси се извршавају на *DNS* серверима.

Уколико се жели видети интернет адреса симболичке адресе, користи се метода `getByName()`.

```
/* Primer Mreza2: Prikazati IP adresu masine koja ima simbolicku adresu "java.fon.bg.ac.yu" .
*/
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.net.*;
```

```

class Mreza2 {
    public static void main(String args[]) throws UnknownHostException {
        InetAddress tekucaAdresa = InetAddress.getByName("java.fon.bg.ac.yu");
        System.out.println(tekucaAdresa); // Prikazuje simbolicku i IP adresu.
        System.out.println(tekucaAdresa.getHostAddress()); // Prikazuje simbolicku adresu.
        System.out.println(tekucaAdresa.getHostName()); // Prikazuje IP adresu.
    }
}
/*
Rezultat:
java.fon.bg.ac.yu/147.91.128.18
147.91.128.18
java.fon.bg.ac.yu
*/

```

Уколико се желе видети све IP адресе које су везане за изабрано симболичко име користи се метода `getAllByName()`.

```

/*
Primer Mreza3: Prikazati IP adrese masina koje su vezane za simbolicku adresu:
"www.nba.com"
*/
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.net.*;
class Mreza3 {
    public static void main(String args[]) throws UnknownHostException {
        InetAddress niziIPAdresa[] = InetAddress.getAllByName("www.cnn.com");
        System.out.println("Broj:" + niziIPAdresa.length);
        for (int i = 0; i < niziIPAdresa.length; i++) {
            System.out.println(niziIPAdresa[i]);
        } // Prikazuje IP adrese racunara.
    }
}
/*
Rezultat: Broj:8
www.cnn.com/64.236.91.24
www.cnn.com/64.236.16.20
www.cnn.com/64.236.16.52
www.cnn.com/64.236.24.12
www.cnn.com/64.236.29.120
www.cnn.com/64.236.91.21
www.cnn.com/64.236.91.22
www.cnn.com/64.236.91.23
*/

```

Задатак MP31: Написати програм који ће преко стандардног улаза да прихвати произвољну симболичку адресу рачунара. Приказати IP адресе задате симболичке адресе.

Задатак MP32: Написати програм који ће преко стандардног улаза да прихвати низ симболичких адреса рачунара. Одредите за сваку симболичку адресу IP адресе. Приказати симболичке адресе и њихове IP адресе у сортираном редоследу (симболичке адресе које имају више IP адреса биће прве приказане).

3.2 URL адреса

Интернет и симболичке адресе рачунара омогућавају да се преко њих приступи до жељених рачунара у мрежи. Уколико се жели приступити до одређених сервиса и датотека на рачунару користи се **URL (Uniform Resource Locator)** адреса.

URL адреса састоји се из четири дела:

Протокол који се користи (**http, ftp, gopher** или **file**), који је одвојен : од остатка адресе. У последње време углавном се ради преко http протокола.

Адреса рачунара (интернет или симболичка адреса). Испред адресе се ставља ‘//’ а на крај адресе се ставља ‘/’ уколико не постоји порт, односно ‘:’ ако постоји.
Број порта (прикључка). Из порта се ставља ‘/’. Овај део није обавезан.
Путања до датотеке, укључујући и име датотеке.

Следећи пример показује све делове задате *URL* адресе.

```
/* Programske zahteve Mreza4: Prikazati svaki od delova URL adrese. */
/*
@uthor Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

import java.net.*;

class Mreza4 {
    public static void main(String args[]) throws MalformedURLException {
        URL hp = new URL("http://java.fon.bg.ac.yu:80/index.htm");
        System.out.println("Protokol:" + hp.getProtocol());
        System.out.println("Port:" + hp.getPort());
        System.out.println("Racunar:" + hp.getHost());
        System.out.println("Datoteka:" + hp.getFile());
        System.out.println("Zajedno:" + hp.toExternalForm());
    }
}
/*
Rezultat:
Protokol: http
Port: 80
Racunar: java.fon.bg.ac.yu
Datoteka: /index.html
Zajedno: http://java.fon.bg.ac.yu:80/index.htm
*/
```

Уколико се жели прочитати произвољна датотека на задатој *URL* адреси, користи се класа `URLConnection`. Следећи пример показује како се користи `URLConnection` класа.

```
/* Primer Mreza5: Prikazati sadrzaj datoteke index.html kojoj se pristupa pomocu URL
adrese.
*/
/*
@uthor Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

import java.net.*;
import java.io.*;
import java.util.Date;

class Mreza5 {

    public static void main(String args[]) throws Exception {
        int c;
        URL url = new URL("http://java.fon.bg.ac.yu/index.htm");
        URLConnection urlc = url.openConnection();
        InputStream is = urlc.getInputStream();
        System.out.println("Datum:" + new Date(urlc.getDate()));
        System.out.println("Vrsta sadrzaja:" + urlc.getContentType());
        System.out.println("Rok trajanja:" + urlc.getExpiration());
        System.out.println("Vreme zadnje izmene:" + new Date(urlc.getLastModified()));
        while (((c = is.read()) != -1)) {
            System.out.println((char) c);
        }
        is.close();
    }
}
```

Задатак МР33: Успоставити везу са произвољном *URL* адресом на којој се налази произвољна датотека. Прочитати датотеку и запамтити њен садржај. Запамћени садржај претражити по задатој речи. Приказати редне бројеве позиција задате речи и укупан број њених појављивања.

3.3 Сокети

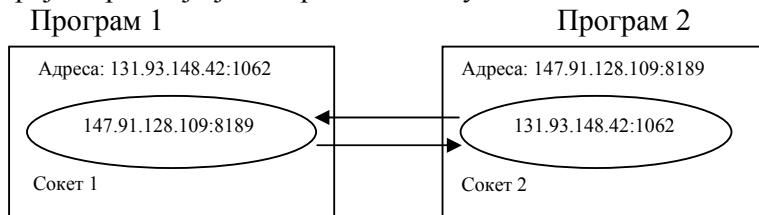
Сокет, у ширем смислу, је механизам који омогућава комуникацију између програма који се извршавају на различитим рачунарима у мрежи. Он је пројектован тако да подржи имплементацију клијент/серверских апликација. Сокети користе *TCP/IP* протокол при повезивању, контроли и преносу података између два или више програма

При повезивању два програма преко сокета, по један сокет се генерише за сваки програм. Сваки од сокета садржи референцу на други сокет. То практично значи да први сокет садржи референцу на други сокет, док други сокет садржи референцу на први сокет.

3.3.1 Адреса сокета

састоји се из два дела:

- адресе рачунара на коме се налази програм који је генерисао сокет
- броја порта који је генерисан помоћу сокета



3.3.2 Конекција између два програма је остварена када се успостави веза између њихових сокета. Сокети, у ужем смислу, представљају објекте помоћу којих се шаљу/прихватају подаци ка/од других сокета.

Сокет је по својој природи улазно-излазни ток и он се понаша на сличан начин као:

- системски објекат *System.in*, помоћу кога се подаци прихватају са стандардног улаза (тастатуре), док се код сокета подаци прихватају са спољашњег улаза (са мреже) од другог сокета.
- системски објекат *System.out*, помоћу кога се подаци шаљу ка стандардном излазу (екрану), док се код сокета подаци шаљу ка спољашњем излазу (ка мрежи) до другог сокета.

Сокети се повезују са улазно-излазним токовима на сличан начин као што је то случај са *System.in* и *System.out* објектима, када се жели извршити обрада података коју сокети разменују.

Типичан сценарио **развоја клијент/серверских апликација** помоћу сокета је следећи:

Покреће се програм на серверском рачунару. Интернет адреса сервера рачунара је 147.91.128.109.

Коришћењем наредбе:

```
ServerSocket ss = new ServerSocket(8189);
```

прави се сервер сокет који се повезује¹⁸ нпр. са портом 8189¹⁹.
Адреса сервер сокета ss је 147.91.128.109:8189.

¹⁸ За сво време трајања програма сокет ослушкује (прати) рад наведеног порта.

¹⁹ Порт 8189 не користи ни један од стандарних сервиса.

Након тога извршава се наредба

```
Socket soketS = ss.accept();
```

којом сервер сокет долази у стање чекања на клијенте. Он ће бити у том стању све док се клијент не повеже са њим.

На клијентској страни се извршава наредба:

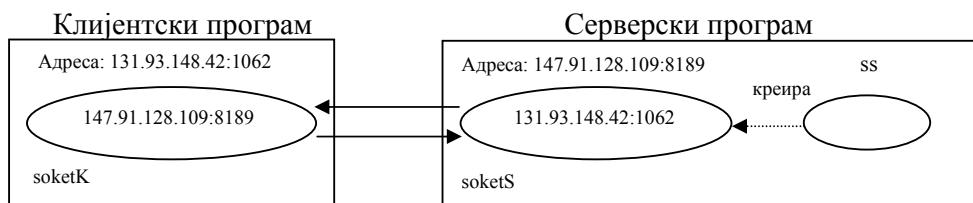
```
Socket soketK = new Socket("147.91.128.109", 8189);
```

којом се врши повезивање клијента са сервером²⁰. Сокет који је направљен на клијентској страни прослеђује до серверског сокета своју адресу: 131.93.148.42:1062. У току успостављања везе (конекције) између серверског сокета и клијентског сокета дешавају се следеће активности:

Клијентски сокет `soketK` добија референцу (`147.91.128.109", 8189`) на серверски сокет.

Серверски сокет се генерише нови сокет `soketS`. Адреса новог сокета ће бити иста као и адреса серверског сокета.

Нови сокет `soketS` добија референцу (`131.93.148.42:1062`) на клијентски сокет.



Као резултат наведених активности добијени су сокети `soketS` и `soketK` који показују један на другог. Пошто се жели извршити размена података између сокета, тако да исти могу да се обраде преко стандардних улазно-излазних уређаја, сокети се повезују са улазно-излазним објектима на следећи начин:

```
BufferedReader in = new BufferedReader(new InputStreamReader(X.getInputStream()));
PrintWriter out = new PrintWriter(X.getOutputStream(), true);
```

`X ∈ (soketS, soketK)`

Након тога је могуће да клијент пошаље поруку до сервера и обрнуто са наредбом:

```
out.println(" Y je spreman za rad\n");
```

`Y ∈ (KLJENT, SERVER)`

Клијент, односно сервер приhvата податке на следећи начин:

```
String line = in.readLine();
```

На крају клијент, односно сервер на стандардном излазу приказују поруку коју је примио:

```
System.out.println(" Z je primio poruku od Z1:" + line);
Z ≠ Z1
Z , Z1 ∈ (KLJENT, SERVER)
```

Из наведеног се може закључити да постоји симетрија метода код размене података између сокета. То значи да место креирања сокета (клијентски или серверски програм), након успостављања конекције, не утиче ни на који начин на размену података. Сокети су равноправни у комуникацији.

Серверски и клијентски програм за наведени пример имају следећи изглед:

²⁰ Сокет на клијентској страни се повезује са сервер сокетом на серверској страни.

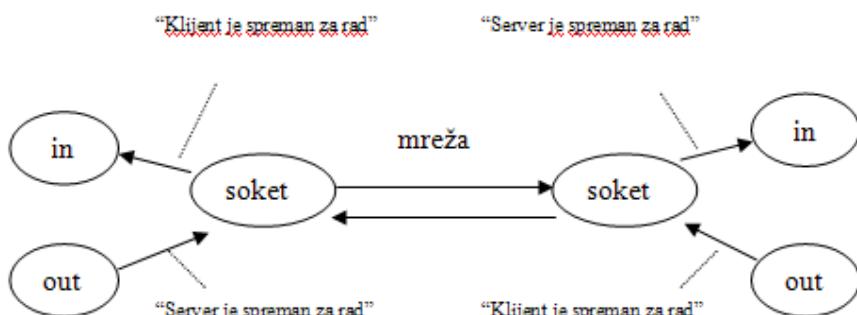
Серверски програм

```
/*
 * Primer MR6S: Napisati program koji ce kreirati serverski soket na portu 8189.
 * Nakon toga se povezati sa klijentskim soketom. Na kraju poslati poruku klijentskom
 * soketu.
 */

/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.io.*;
import java.net.*;
public class ServerSoket {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(8189);
            Socket soketS = ss.accept();
            BufferedReader in = new BufferedReader(new InputStreamReader(soketS.getInputStream()));
            PrintWriter out = new PrintWriter(soketS.getOutputStream(), true);
            out.println(" SERVER je spreman za rad\n");
            String line = in.readLine();
            System.out.println(" SERVER je primio poruku od klijenta:" + line);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Клијентски програм

```
/*
 * Primer MR6K: Napisati program koji ce kreirati klijentski soket, koji ce se
 * povezati sa serverskim soketom koji je podignut na racunaru cija je adresa
 * 147.91.128.109 na portu 8189. Poslati poruku serverom racunaru.
 */
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.io.*;
import java.net.*;
public class SoketKlijent {
    public static void main(String[] args) {
        try {
            Socket soketK = new Socket("147.91.128.109", 8189);
            BufferedReader in = new BufferedReader(new InputStreamReader(soketK.getInputStream()));
            PrintWriter out = new PrintWriter(soketK.getOutputStream(), true);
            out.println(" KLIJENT je spreman za rad\n");
            String line = in.readLine();
            System.out.println(" KLIJENT je primio poruku od servera:" + line);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```



Уколико се жели видети:

- порт сокета на који показује други сокет, користи се метода `getPort()`.
- IP адреса сокета на који показује други сокет, користи се метода `getInetAddress()`.
- локални порт на коме је подигнут сокет користи се метода `getLocalPort()`. То значи да се пуне адресе сокета на који показује други сокет добија као: `getInetAddress() + getPort()` што се може видети у следећем примеру:

Серверски програм

```
/*
Primer MR7S: Prikazati IP adresu racunara i broj porta na kome se nalazi
klijentski soket. Na kraju prikazati broj porta na kome se nalazi serverski soket.
*/
/*
@auther Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

import java.io.*;
import java.net.*;

public class ServerSoket {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(8189);
            System.out.println("SERVER");
            Socket soketS = ss.accept();
            // ia dobija IP adresu racunara na kome se nalazi klijentski soket
            InetAddress ia = soketS.getInetAddress();
            // getPort() metoda prikazuje port na kome se nalazi klijentski soket
            System.out.println(ia + " " + soketS.getPort());
            // getLocalPort() metoda prikazuje port na kome se nalazi serverski soket (8189)
            System.out.println(soketS.getLocalPort());
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Клијентски програм за наведени пример је:

```
/*
MR7K: Napisati program koji ce kreirati klijentski soket koji ce se
povezati sa serverskim soketom koji je podignut na lokalnom racunatu na portu 8189.
Prikazati IP adresu racunara i broj porta na kome se nalazi serverski soket.
Na kraju prikazati broj porta na kome se nalazi klijentski soket.
*/
/*
@auther Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

import java.io.*;
import java.net.*;

public class SoketKlijent {
    public static void main(String[] args) {
        try {
            String s;
            Socket soketK = new Socket("147.91.128.109", 8189);
            InetAddress ia = soketK.getInetAddress();
            System.out.println(ia + " " + soketK.getPort() + " " + soketK.getLocalPort());
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Задатак МР34: Направити серверски и клијентски сокет који међусобно разменују поруке све док се не искљуци један од сокета (*chat* 2 сокета). Нацртати изглед оперативне меморије.

3.3.3 Повезивање сервера са више клијената

Сокети омогућавају да се више клијентских програма (клијент сокета) повеже на један серверски програм (серверски сокет). За сваки од клијентских сокета прави се по једна нит, тако да се у оквиру серверског програма конкурентно извршава више нити. Наведене нити могу да приступе заједничким ресурсима сервера.

У следећа 2 примера, више клијената повезаће се са серверским програмом. У току извршења, наведени клијенти ће приступати и обрађивати заједнички атрибут *Kolicina* серверског програма. У првом примеру повезаће се сервер са више telnet програма (клијенти), док ће у другом примеру сервер да се повеже са Јава клијентима, односно програмима који су направљени у Јави.

Server и Telnet програми (клијенти)

```
/*
Primer MR8S: Napisati program koji ce kreirati serverski soket na portu 8189. Serverski
soket moze da se poveze sa najvise 10 klijenata (klijentskih soketa). Za svakog
klijenta napraviti posebnu nit koja ce se nezavisno izvrsavati u odnosu na druge niti.
U okviru svake niti ce se vrsiti obrada kolicine robe (prodaja i nabavka).
Kolicina robe ce biti zajednicki atribut svih klijenata.
*/

/*
@auther Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

import java.io.*;
import java.net.*;
public class ObradaRobe {
    public static void main(String[] args) {
        try {
            KreiranjeNiti kn = new KreiranjeNiti();
            kn.Kreiranje();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
    class KreiranjeNiti {
        int kolicina;
        ObradaNiti on[];
        ServerSocket ss;

        KreiranjeNiti() {
            on = new ObradaNiti[10];
        }
        public void Kreiranje() {
            try {
                ss = new ServerSocket(8189);
                kolicina = 10;
                for (int brojKlijenta = 0; brojKlijenta < 10; brojKlijenta++) {
                    Socket soketS = ss.accept();
                    System.out.println("Klijent " + brojKlijenta);
                    on[brojKlijenta] = new ObradaNiti(soketS, brojKlijenta, this);
                    on[brojKlijenta].start();
                }
            } catch (Exception e) {
                System.out.println(e + " greska!");
            }
        }
    }
}
```

```

class ObradaNiti extends Thread {
    public ObradaNiti(Socket soketS1, int c, KreiranjeNiti kn1) {
        soketS = soketS1;
        brojKlijenta = c + 1;
        kn = kn1;
    }
    public void run() {
        try {
            in = new BufferedReader(new InputStreamReader(soketS.getInputStream()));
            out = new PrintWriter(soketS.getOutputStream(), true);
            boolean done = false;
            while (!done) {
                out.println("Izaberite jednu od sledečih opcija:\n");
                out.println("1. PRODAJA. 2. NABAVKA 3. IZLAZ");
                out.println(" ");
                String line = in.readLine();
                if (line.equals("")) {
                    done = true;
                } else {
                    switch (line.charAt(0)) {
                        case '1':
                            out.println("Klijent: (" + brojKlijenta + "): IZABRANA PRODAJA");
                            if (kn.kolicina == 0) {
                                out.println("Nema robe na zalihamu!");
                            } else {
                                kn.kolicina = kn.kolicina - 1;
                            }
                            break;
                        case '2':
                            out.println("Klijent: (" + brojKlijenta + "): IZABRANA NABAVKA");
                            kn.kolicina = kn.kolicina + 1;
                            break;
                        default:
                            done = true;
                    }
                }
                out.println("Ukupno je ostalo komada:" + kn.kolicina);
            }
            out.println("999");
            soketS.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
    private Socket soketS;
    private int brojKlijenta;
    private KreiranjeNiti kn;
    BufferedReader in;
    PrintWriter out;
}

```

Када се позове *telnet* програм, који је клијентски програм појавиће се:

Microsoft Telnet >

Повезивање са серверским програмом се ради преко наредбе:

Open 127.0.0.1 8189

Детаљни опис програма MR8S:

На серверској страни коришћењем наредбе:

```
ServerSocket ss = new ServerSocket(8189);
```

прави се сервер сокет који се повезује²¹ са портом 8189²².

Након тога извршава се наредба

```
Socket soketK = ss.accept();
```

којом сервер сокет долази у стање чекања на клијенте. Он ће бити у том стању све док се клијент не повеже преко одговарајуће IP адресе и порта 8189 са њим.

²¹ За сво време трајања програма сокет ослушкује (прати) рад наведеног порта.

²² Порт 8189 не користи ни један од стандарних сервиса.

На клијентској страни (телнет програм) се извршава наредба:

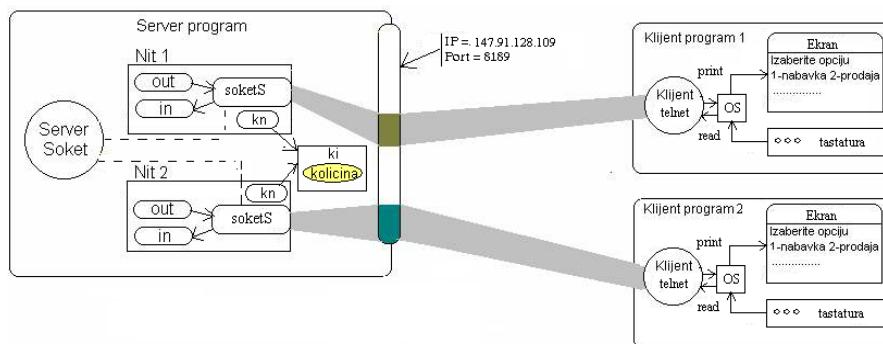
```
Socket socket = new Socket(147.91.128.109,8189) ;
```

којом се врши повезивање клијента са сервером. Адреса сокета који је направљен на клијентској страни (преко наредбе `new Socket(147.91.128.109, 8189)`), се прослеђује до сервера.

Након тога извршава се наредба:

```
on[brojKlijenta] = new ObradaNiti(soketS, brojKlijenta, this);
```

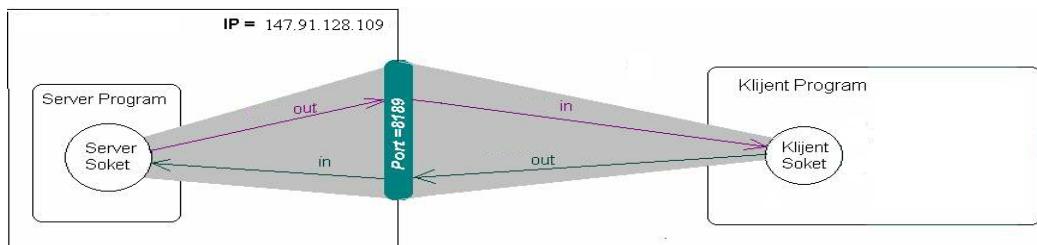
која за сваки позив клијента прави нит, коме се као аргументи конструктора шаљу: сокет `SoketS` (садржи адресу клијентског сокета), који је генерирао серверски сокет `ss`, број клијента који је добијен по редоследу повезивања клијената са сервером и објекат типа `KreiranjeNiti` који контролише извршење комплетног програма. Новокреирана нит се чува у атрибуту `on[brojKlijenta]`²³ и она је одговорна за комуникацију између сервера и клијента.



Новокреирана нит покреће методу `start()` која врши иницијализацију нити. Након иницијализације метода `start()` покреће методу `run()`. У оквиру методе `run()` креирају се објекти `in` (класе `BufferedReader`) и `out` (класе `PrintWriter`) који претстављају улазне, односно излазне токове сокета²⁴:

```
BufferedReader in = new BufferedReader(new InputStreamReader(soketS.getInputStream()));
PrintWriter out = new PrintWriter(soketS.getOutputStream(),true);
```

Све што се пошаље преко излазног тока сервера постаје улазни ток за клијента. Такође сви излазни токови од клијента постају улазни токови за сервер



У нашем примеру клијент је био програм **telnet** помоћу кога смо се повезали са сервером.

Наредбе као што су:

```
out.println("Izaberite jednu od sledećih opcija:\n");
out.println("1.PRODAJA. 2.NABAVKA 3. IZLAZ");
...
out.println(...)
```

²³ При чему атрибут претставља низ могућих нити. Свака нит је везана за једног клијента

²⁴ Помоћу наведених токова се врши двосмерна комуникација сервера и клијента.

приказивале су наведене садржаје на екрану преко telnet телнет програма.

Наредба `out.println()` сервера шаље податке до клијента (*telnet*) .

Са друге стране нареба `in.readLine()` сервера прихвата податке од клијента.

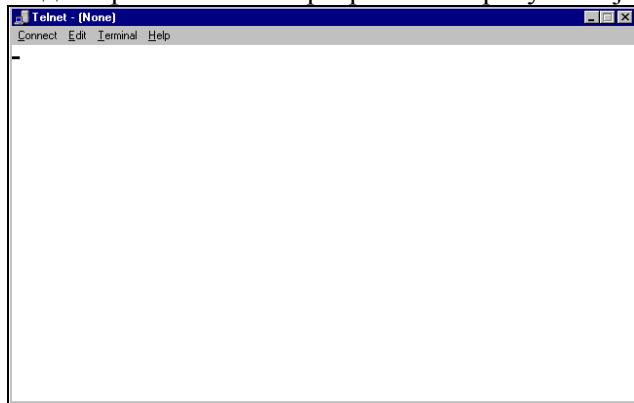
При извршењу наредбе `run()`, свака од креираних нити чува референцу (атрибут `kn` који је добио `this`) према објекту који садржи атрибут (`Kolicina`), који је дељив за сваку од креираних нити.

За излаз из нити може се користити метода `stop()` али се иста сматра застарелом (она може да доведе до проблема у раду оперативног система). За излаз из нити најбоље је изаћи из `run()` методе, што је у нашем случају избор опције 3, која ће да доведе до прекида рада клијента . На крају рада са сокетом позива се метода `close()` .

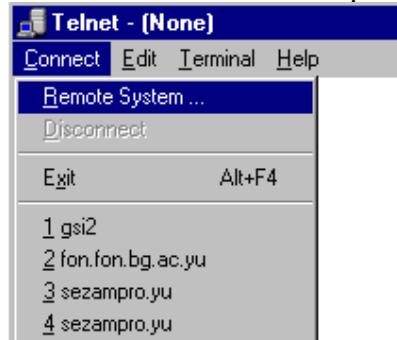
Кораци код рада са програмом MR8S:

1. Отворите едитор
2. Укуцајте програмски код за серверски сокет
3. Сачувавјте код - “ime_dat.java”
4. Компајлирате код - “ime_dat.class”
5. Извршите програм - серверски сокет
6. Покрените телнет програме да симулирајте клијентске сокете

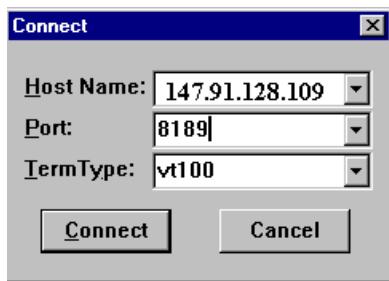
Код покретања *telnet* програма на екрану се појављује следеће:



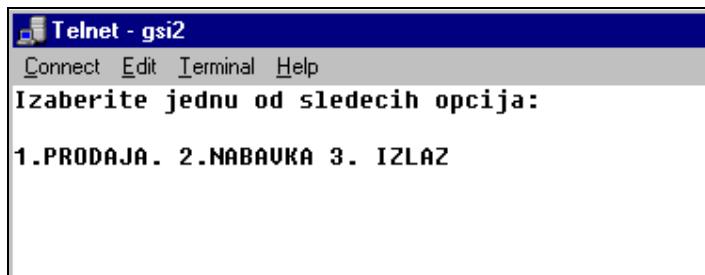
Након тога позовите изаберите опцију *Connect-Remote System*



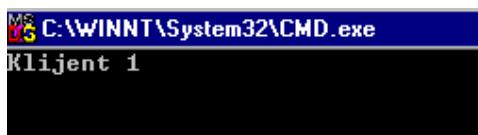
На екрану се појављује:



Изаберите IP рачунара (у нашем случају то је 147.91.128.109) и број порта 8189. Након тога, уколико је успешно извршено логовање појавиће се:



На серверском програму, након повезивања клијента са њим појавиће се:



То значи да се први клијент повезао са серверским програмом.

На сличан начин можете да повежете друге клијенте са серверским програмом.

Суштина наведеног програма своди се на то да више клијената користи исти серверски програм, који ради над дељивим атрибутом kolicina. То практично значи да сваки клијент може да повећава, односно смањује заједнички атрибут kolicina, коришћењем опција (1. PRODAJA, 2. NABAUKA).

Задатак MP35: Допуните наведени пример тако да серверски програм јави свим клијентима (који су са њим повезани) да је атрибут kolicina добио вредност 0.

Сервер и Јава програми (клијенти)

У случају када је сервер повезан са више Јава програма (клијената), клијентски програм је следећи (серверски програм је исти као и програм MR8S):

```
/*
Primer MR9K: Napisati program koji ce kreirati klijentski soket koji ce se
povezati sa serverskim soketom koji je podignut na racunaru cija je IP adresa
147.91.128.109 na portu 8189.
*/
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.io.*;
```

```

import java.net.*;
public class SoketKlijent1 {
    public static void main(String[] args) {
        try {
            String s, line;
            Socket soketK = new Socket("147.91.128.109", 8189);
            BufferedReader in = new BufferedReader(new InputStreamReader(soketK.getInputStream()));
            PrintWriter out = new PrintWriter(soketK.getOutputStream(), true);
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            while (true) {
                line = in.readLine();
                // Kada serverski program posalje "999" prekinuce se izvrsenje klijenta.
                if (line.equals("999")) {
                    soketK.close();
                    break;
                }
                System.out.println(line);
                // Kada serverski program posalje " ", klijent dobija mogucnost da izabere opciju.
                if (line.equals(" ")) {
                    s = br.readLine();
                    out.println(s);
                }
            } //while
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

Уколико се јави потреба да серверски програм пошаље клијентима обавештење, уколико вредност количине падне на 0, тада серверски програм има следећи изглед:

```

/*
Primer MR10S: Napisati program koji ce kreirati serverski soket na portu 8189. Serverski
soket moze da se poveze sa najvise 10 klijenata (klijentskih soketa). Za svakog
klijenta napraviti posebnu nit koja ce se nezavisno izvrsavati u odnosu na druge niti.
U okviru svake niti ce se vrsiti obrada koliceine robe (prodaja i nabavka).
Kolicina robe ce biti zajednicki atribut svih klijenata.
Kada roba padne na kolicinu jednaku 0 server treba da o tome obavesti sve klijente.
*/
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.io.*;
import java.net.*;

public class ObradaRobe1 {
    public static void main(String[] args) {
        try {
            KreiranjeNiti kn = new KreiranjeNiti();
            kn.Kreiranje();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
class KreiranjeNiti {
    int kolicina;
    ObradaNiti on[];
    ServerSocket ss;
    int brojKlijenta;

    KreiranjeNiti() {
        on = new ObradaNiti[10];
    }
    public void Kreiranje() {
        try {
            ss = new ServerSocket(8189);
            kolicina = 10;
            for (brojKlijenta = 0; brojKlijenta < 10; brojKlijenta++) {
                Socket soketsS = ss.accept();
                System.out.println("Klijent " + brojKlijenta);
                on[brojKlijenta] = new ObradaNiti(soketsS, brojKlijenta, this);
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
class ObradaNiti {
    ...
}

```

```

        on[brojKlijenta].start();
    }
    } catch (Exception e) {
        System.out.println(e + " greska!");
    }
}
public void Prodaja(float p) {
    kolicina -= p;
    if (kolicina <= 0) {
        Azuriranje();
    }
}
public void Nabavka(float n) {
    kolicina += n;
}

public void Azuriranje() {
    NitObavestenje obavestenje = new NitObavestenje(this, "Magacin se upravo izpraznio !");
    obavestenje.start();
    System.out.println("kreirao nit obavestenje");
}
}

class ObradaNiti extends Thread {
    public ObradaNiti(Socket soketS1, int c, KreiranjeNiti kn1) {
        soketS = soketS1;
        brojKlijenta = c + 1;
        kn = kn1;
    }
    public void run() {
        try {
            in = new BufferedReader(new InputStreamReader(soketS.getInputStream()));
            out = new PrintWriter(soketS.getOutputStream(), true);
            boolean done = false;
            while (!done) {
                out.println("Izaberite jednu od sledecih opcija:\n");
                out.println("1.PRODAJA 2.NABAVKA 3. IZLAZ");
                out.println(" ");
                String line = in.readLine();
                if (line.equals("")) {
                    out.println("Zavrsetak rada klijenta");
                    out.println("999");
                    done = true;
                } else {
                    switch (line.charAt(0)) {
                        case '1':
                            out.println("Echo: (" + brojKlijenta + "): IZABRANA PRODAJA");
                            if (kn.kolicina - 4 < 0) {
                                out.println("Nema dovoljno robe na zalihama da bi se izvršila prodaja!");
                            }
                            else {
                                kn.Prodaja(4);
                            }
                            break;
                        case '2':
                            out.println("Echo: (" + brojKlijenta + "): IZABRANA NABAVKA");
                            kn.Nabavka(2);
                            break;
                        case '3':
                            out.println("Zavrsetak rada klijenta");
                            out.println("999");
                            done = true;
                    }
                }
                out.println("Ukupno je ostalo komada:" + kn.kolicina);
            }
            soketS.close();
            System.out.println("Zatvorio klijenta " + brojKlijenta);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
    private Socket soketS;
    public int brojKlijenta;
    private KreiranjeNiti kn;
    BufferedReader in;
    PrintWriter out;
}
}

```

```

class NitObavestenje extends Thread {
    public NitObavestenje(KreiranjeNiti k, String o) {
        kn = k;
        obavestenje = o;
    }
    private KreiranjeNiti kn;
    private String obavestenje;

    public void run() {
        for (int i = 0; i < kn.brojKlijenta; i++) {
            try {
                kn.on[i].out.println("Echo(" + kn.on[i].brojKlijenta + ") " + obavestenje);
                System.out.println("Obavestio klijenta " + (i + 1));
            } catch (Exception e) {
                System.out.println("Nema klijenta " + (i + 1));
            }
        }
    }
}

```

Клијент има две нити, главну која ће да шаље податке до сервера, док ће друга нит да прихвата обавештење од сервера.

```

/*
Primer MR10K: Napisati program koji ce kreirati klijentski soket koji ce se
povezati sa serverskim soketom koji je podignut na racunatu cija je IP adresa
127.0.0.1 na portu 8189. Omoguciti da klijent moze u svakom trenutku da primi
poruku od servera.
*/
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.io.*;
import java.net.*;

public class SoketKlijent1 {
    public static void main(String[] args) {
        try {
            String s;
            Socket soketK = new Socket("127.0.0.1", 8189);
            BufferedReader in = new BufferedReader(new InputStreamReader(soketK.getInputStream()));
            PrintWriter out = new PrintWriter(soketK.getOutputStream(), true);
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            boolean signal = true;
            NitKlijent nk = new NitKlijent(in);
            nk.start();
            while (true) { // Prihvata preko tastature podatke
                s = br.readLine();
                // Salje podatke do servera
                out.println(s);
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

class NitKlijent extends Thread {

    NitKlijent(BufferedReader in1) {
        in = in1;
        signal = true;
    }

    public void run() {
        try {
            while (signal) { // Prihvata podatke od servera
                String line = in.readLine();
                if (line.equals("999")) {
                    break;
                }
                // Prikazuje podatke na monitoru
                System.out.println(line);
            }
        } catch (Exception e) {

```

```
        System.out.println("Lose primljena poruka od servera!");
    }
}
void Prekini() {
    signal = false;
}
boolean signal = true;
BufferedReader in;
}
```

Задатак MP36: Написати сервер који ће прекинути рад клијента у случају да клијент посаље поруку серверу да се њему (клијенту) прекине рад.

Вежба BMP31: Нацртати изглед оперативне меморије за програме PR10S и PR10K.

Уколико постоји потреба да сервер посредује у комуникацији више клијената (*chat* програм), дајемо следећи пример:

```

/*Primer MR11S: Napraviti serverski soket koji ce da posreduju u
 komunikaciju izmedju najvise 10 ucesnika
*/
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

import java.io.*;
import java.net.*;

public class Chat {
    public static void main(String[] args) {
        try {
            PovezivanjeSaUcesnicima psu = new PovezivanjeSaUcesnicima();
            psu.Kreiranje();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

class PovezivanjeSaUcesnicima {

    int brojUcesnika;
    NitUcesnika[] np = new NitUcesnika[10]; // Moze najvise 10 ucesnika da se poveze sa
    // serverom.
    public void Kreiranje() {
        System.out.println("SERVER JE SPREMAN ZA RAD!!!\n");
        try {
            ServerSocket ss = new ServerSocket(8189);
            for (brojUcesnika = 0; brojUcesnika < 10; brojUcesnika++) {
                Socket soketS = ss.accept();
                System.out.println("Klijent " + (brojUcesnika + 1));
                np[brojUcesnika] = new NitUcesnika(soketS, brojUcesnika, this);
                np[brojUcesnika].start();
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    public void PosaljiSvimaPoruku(String poruka, int bklijenta) {
        NitObavestenje obavestenje = new NitObavestenje(this, poruka, bklijenta);
        obavestenje.start();
    }
}

class NitUcesnika extends Thread {

    public NitUcesnika(Socket soketS1, int brojUcesnika1, PovezivanjeSaUcesnicima psu1) {
        soketS = soketS1;
        brojUcesnika = brojUcesnika1 + 1;
        psu = psu1;
        try {
            in = new BufferedReader(new InputStreamReader(soketS.getInputStream()));
            out = new PrintWriter(soketS.getOutputStream(), true);
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    public void run() {
        try {
            out.println("SERVER U PRIPREMI POSREDOVANJA U RAZGOVORU:\n");
            out.println("Unesi svoje ime:\n");
            ime = in.readLine();
            out.println("Uneto je ime:\n\n" + ime);
            out.println("SERVER JE SPREMAN ZA POSREDOVANJA U RAZGOVORU:\n");
            while (true) {
                String line = in.readLine();
                psu.PosaljiSvimaPoruku(line, brojUcesnika);
            }
        }
    }
}

```

```

        } catch (Exception e) {
            System.out.println(e);
        }
    }
    public Socket soketS;
    public int brojUcesnika;
    private PovezivanjeSaUcesnicima psu;
    BufferedReader in;
    PrintWriter out;
    String ime;
}

class NitObavestenje extends Thread {

    public NitObavestenje(PovezivanjeSaUcesnicima psu, String o, int bucesnikal) {
        psu = psu;
        obavestenje = o;
        bucesnika = bucesnikal - 1;
    }
    private String obavestenje;

    public void run() {
        for (int i = 0; i < psu.brojUcesnika; i++) {
            try {
                psu.np[i].out.println(psu.np[bucesnika].ime + " :" + obavestenje);
            } catch (Exception e) {
                System.out.println("Nema klijenta " + (i + 1));
            }
        }
    }
}

private PovezivanjeSaUcesnicima psu;
int bucesnika;
}

```

```

/*
Primer MR11K: Napisati program koji ce kreirati klijentski soket koji ce se
povezati sa serverskim soketom koji je podignut na racunatu cija je IP adresa
127.0.0.1 na portu 8189. Serverski soket treba da omoguci menjusobnu razmenu
poruka vise ucesnika u razgovoru.
*/

/*
@auther Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

import java.io.*;
import java.net.*;

public class SoketKlijent1 {

    public static void main(String[] args) {
        try {
            String s;
            Socket soketK = new Socket("127.0.0.1", 8189);
            BufferedReader in = new BufferedReader(new InputStreamReader(soketK.getInputStream()));
            PrintWriter out = new PrintWriter(soketK.getOutputStream(), true);
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            boolean signal = true;
            NitKlijent nk = new NitKlijent(in);
            nk.start();
            s = br.readLine();
            out.println(s);
            while (true) {
                s = br.readLine();
                out.println(s);
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

```

class NitKlijent extends Thread {
    NitKlijent(BufferedReader in1) {
        in = in1;
        signal = true;
    }
    public void run() {
        try {
            while (signal) {
                String line = in.readLine();
                System.out.println(line);
            }
        } catch (Exception e) {
            System.out.println("Lose primeljena poruka od servera!");
        }
    }
    //void Prekini() { signal = false;}
    boolean signal = true;
    BufferedReader in;
}

```

Задатак MP7: Написати сервер програм који ће да прати број порука који је сваки од клијената послao у једној *chat* сесији. Уколико неко од клијената пређе дозвољен број порука упозорава се да више не може да шаље поруке. Уколико клијент након упозорења покуша да пошаље поруку сервер прекида његов рад и о томе обавестава остале клијенте.

Вежба BMP32: Нацртати изглед оперативне меморије за програме PR11S и PR11K.

3.4 Слање електронске поште

Применом концепта сокета, може се омогућити комуникација са *mail* серверима самим тим и слање електронске поште. При размени података са *mail* сервером користи се *smtp* протокол.

SMTP (Simple Mail Transfer Protocol) је стандардни *e-mail* протокол Интернета којим се дефинише формат поруке која се шаље. Код сваког програма за пријем електронске поште увек се дефинише одговарајући *SMTP* сервер.

Комуникација која се успоставља између *mail* сервера и сокета који је повезан са њим се обавља на тај начин да се излазним током од сокета ка *mail* серверу шаљу *smtp* команде, а у супротном смеру, кроз улазни ток сокета долазе одговори севера на те команде.

Основне *SMTP* команде које омогућавају конекцију са *mail* сервером и слање електронске поште су:

1. **MAIL< space>FROM** reverse_path<enter>

Ова команда означава почетак нове *mail* трансакције

reverse_path треба да садржи адресу рачунара пошиљаоца на коју треба да се шаљу поруке

2. **RCPT<space> TO <forward_path>**

Ова команда даје адресу једног примаоца. Она се може поновити произвољан број пута.

3. **DATA<enter>**

Subject:Subject<enter>

From:From<enter>

<enter>

<.>

Овде се уписује текст поруке.Ако желите да прикажете поља *Subject* и *From* можете их дефинисати на горе наведени начин.Крај *DATA* одељка се задаје тачком и то у одвојеном реду.

4.QUIT<enter>

Затварање *SMTP* канала

```
/*
MR12:Napisati program koji ce uspostaviti konekciju sa proizvoljnim mail serverom
i poslati mu poruku.
*/
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

import java.io.*;
import java.net.*;
import java.util.Date;

class SlanjeEMaila {

    InputStream is;
    OutputStream os;
    String status;

    SlanjeEMaila() {
        status = "";
    }

    void Slanje() throws IOException {//povezuje soket sa mail serverom
        Socket socket = new Socket("smtp.fon.bg.ac.yu", 25);
        //izlazni tok iz soketa postaje ulazni tok mail servisa
        is = socket.getInputStream();
        //Ulazni tok soketa postaje izlazni tok mail servisa
        os = socket.getOutputStream();
        SMTPTransakcija();
    }

    void odgovorServera() throws IOException {
        for (long l = System.currentTimeMillis(); System.currentTimeMillis() - l < 10000L;) //Daje se odredjeno vreme za odgovor (10 sekundi)
        {
            status = "";
            status = CitanjeIzToka();
            if (status.length() > 0) {
                System.out.println("ODGOVOR SERVERA: " + status);
            }
        }
    }

    //Metoda koja salje zahteve i prima odgovore od servera

    void SMTPTransakcija() throws IOException {
        Date date = new Date();
        odgovorServera();
        UpisivanjeUTok("HELO 127.0.0.1\r\n");
        odgovorServera();
        UpisivanjeUTok("MAIL FROM: vlajic\r\n");
        odgovorServera();
        UpisivanjeUTok("RCPT TO: vlajic@fon.bg.ac.yu\r\n");
        odgovorServera();
        UpisivanjeUTok("DATA\r\n");
        odgovorServera();
        UpisivanjeUTok("Subject: " + "Test poruka" + "\r\nFrom: Sinisa\r\nTo: ");
        UpisivanjeUTok("vlajic@fon.bg.ac.yu\r\nDate: " + date.toString());
        UpisivanjeUTok("\r\n\r\nDan je lep dan!!!\r\n.\r\n");
        odgovorServera();
        UpisivanjeUTok("QUIT\r\n");
        odgovorServera();
        System.out.println("kraj SMTP transakcije");
    }
}
```

```
void UpisivanjeUTok(String s) throws IOException {
    if (s.length() > 0) {
        for (int i = 0; i < s.length(); i++) {
            os.write(s.charAt(i));
        }
    }
}

String CitanjeIzToka() throws IOException {
    String s;
    for (s = new String(""); is.available() > 0 && s.length() < 255;
         s = s + String.valueOf((char) is.read())) {
    }
    return s;
}

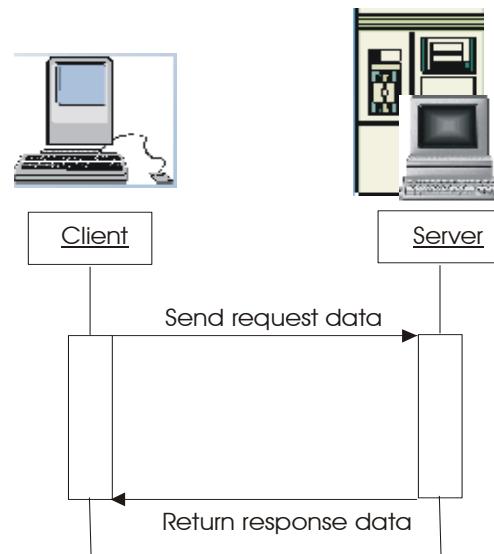
public static void main(String args[]) throws IOException {
    SlanjeEMaila sem = new SlanjeEMaila();
    sem.Slanje();
}
}
```

4. РАД СА УДАЉЕНИМ ОБЈЕКТИМА

Технологија **Позива удаљених метода** (*RMI - Remote Method Invocation*) омогућава да објекат (који је декларисан унутар клијентског програма), који се налази на некој машини у мрежи, може позвати методе удаљеног објекта (који је декларисан унутар серверског програма) који се налази на некој другој машини у мрежи.

4.1 Улоге клијента и сервера

Код традиционалног клијент-сервер модела (*Слика РМИ1*), клијент поставља захтев серверу да изврши неку од његових метода. Захтев се кроз мрежу преноси у одговарајућем формату који је у складу са дефинисаним протоколом преноса података. Сервер анализира захтев, прави одговор и такође га у одговарајућем формату враћа назад до клијента. Клијент приhvата одговор и приказује га крајњем кориснику.



Слика РМИ1 – Слање клијентског захтева до сервера

4.2 Позив удаљених метода

У *RMI* технологији, објекат који позива удаљену методу назива се **клијентски објекат**. Удаљени (remote) објекат се назива **серверски објекат**. Рачунар којим се позива удаљена метода је **клијент** за тај позив, а рачунар са објектом који садржи позвану методу и који обрађује тај позив је **сервер** за тај позив. Сервер може позвати методе неких других удаљених објеката (тада сервер постаје клијент).

4.2.1 Комуникација клијентског и серверског објекта

Када клијентски програм преко клијентског објекта позива методу серверског објекта, он зове обичну Јава методу која се налази у стуб објекту (сурогат серверског објекта) који је на клијентској машини. Када стуб приhvati захтев за извршење неке од метода (која може имати параметре) од клијентског објекта, он врши његову трансформацију у информациони блок који се преноси кроз мрежу до сервера.

Информациони блок се састоји од:

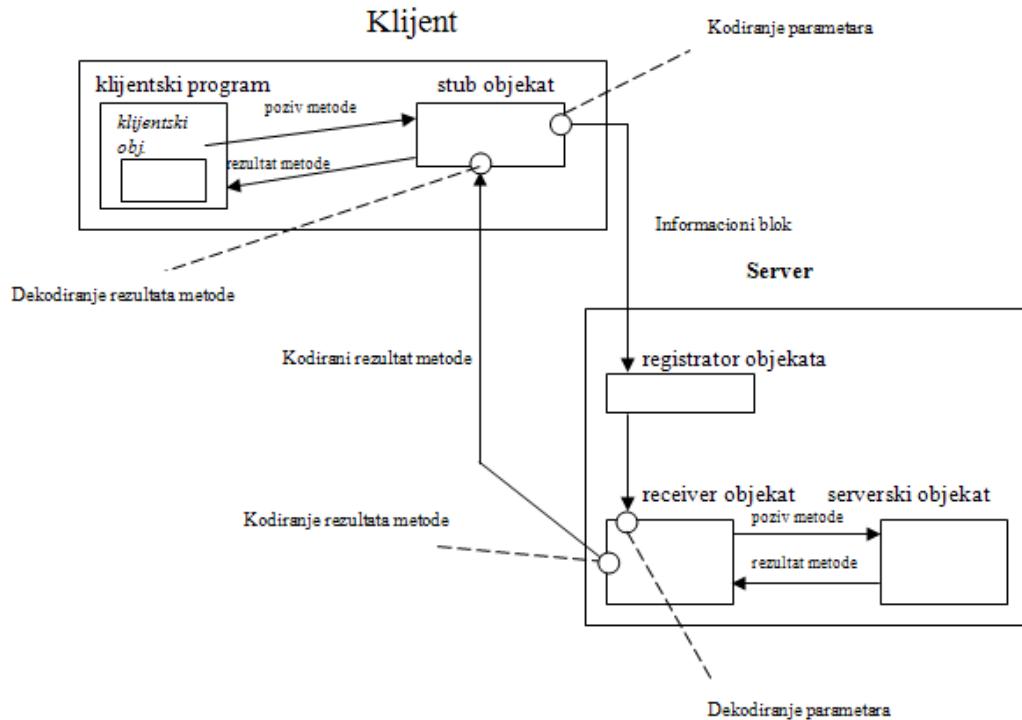
- идентификатора серверског објекта,
- описа методе која се позива и
- кодираних параметара методе који су представљени низом бајтова.

Напомена: Кодирање параметара омогућава механизам **серијализације**. Процес кодирања параметара се назива ***parameter marshaling***. Сврха овог процеса је да конвертује параметре у формат који је погодан за транспорт кроз мрежу од клијентског до серверског програма.

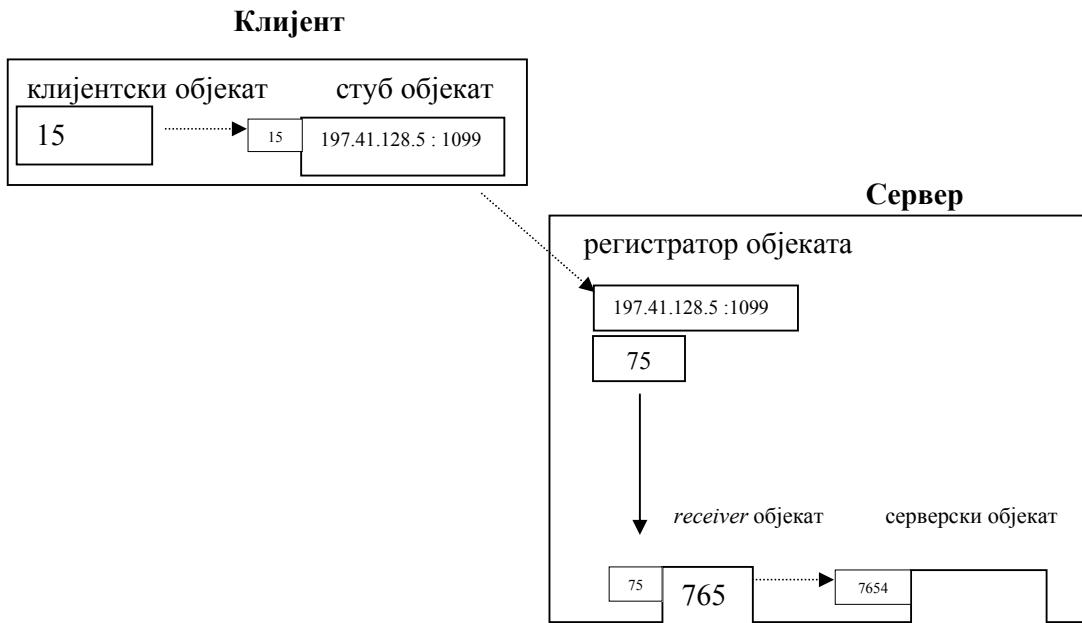
Стуб затим шаље информациони блок до регистратора објекта који је подигнут на серверу. Регистратор објекта проверава да ли постоји позвани серверски објекат. Уколико постоји регистратор објекта преусмерава информациони блок до *receiver* објекта који изводи следеће акције:

- параметри се декодирају (*unmarshalling*),
- лоцира се објекат који је позван,
- позива се жељена метода којој се шаљу декодирани параметри,
- прихватају се и кодирају подаци (*marshalling*) који су настали као резултат извршења методе,
- кодирани подаци се шаљу до стуб објекта који је на клијенту.

На крају стуб врши декодирање (*unmarshalling*) података које је добио од сервера и приказује их крајњем кориснику.



4.2.2 Веза између клијентског и серверског објекта



Клијентски објекат је у суштини објектна променљива која садржи референцу на стуб објекат. Када клијентски објекат позива методу серверског објекта, он прво позива стуб објекат. Стуб објекат садржи IP адресу регистратора објекта коме шаље информациони блок. Када се методе позивају помоћу RMI протокола подразумева се да је регистратор објекта објекта подигнут на порту 1099. Регистратор објекта шаље информациони блок до receiver објекта који га обрађује и позива серверски објекат да изврши методу. Receiver објекат садржи референцу на порт серверског објекта чија се метода позива.

4.2.3 Структура серверског програма

Серверски програм у контексту RMI технологије садржи:

Интерфејс који дефинише скуп операција (метода) које позива клијентски програм. Назив интерфејса представља основу именовања свих осталих класа серверског и клијентског програма (нпр. ако интерфејс има назив `Racunaj`, тада ће класа која имплементира интерфејс добити назив `RacunajImpl`, док ће класа која креира серверски објекат добити назив `RacunajServer`).

Класу (нпр. `RacunajImpl`) која имплементира наведени интерфејс.

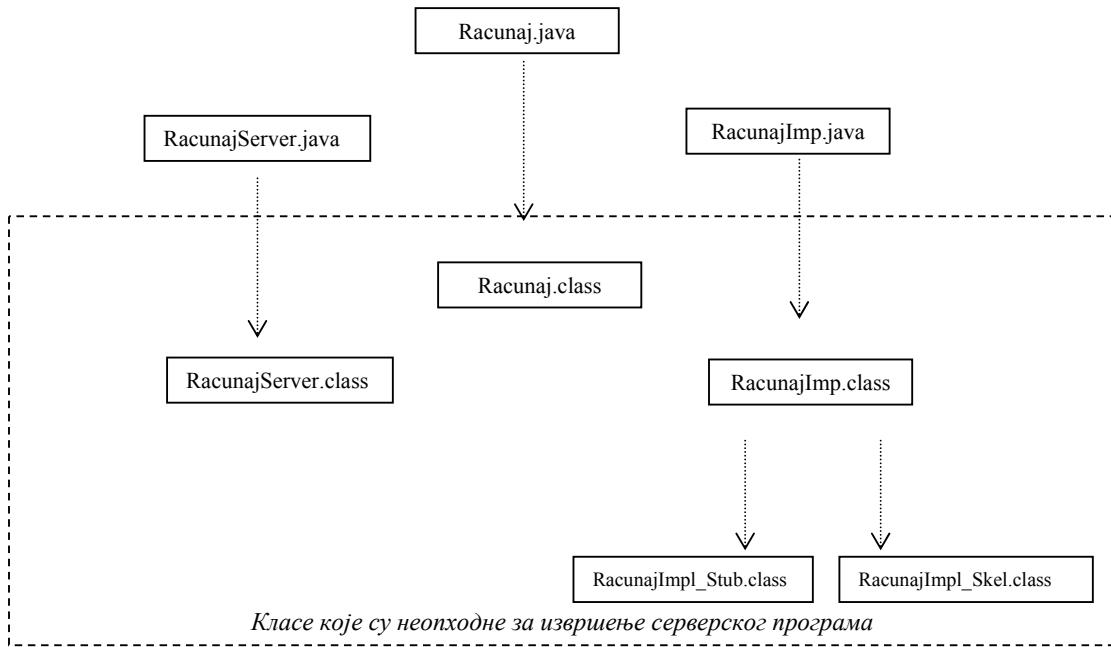
Класу (нпр. `RacunajServer`) која креира серверске објекте.

Када се серверски програм компајлира тада се добијају три class датотеке (нпр. `Racunaj.class`, `RacunajImpl.class` и `RacunajServer.class`).

Поред наведене три class датотеке постоје још две class датотеке (нпр: `RacunajImpl_Stub.class` и `RacunajImpl_Skel.class`²⁵) које се добијају када се позове наредба:

```
rmic RacunajImpl
```

²⁵ Класа `Racunaj_Skel.class` је била потребна код RMI-а за JDK 1.1 верзију. То значи да у новим верзијама Јаве ова класа није потребна за рад RMI-а.



Извршавање серверског програма

ПРЕ извршења серверског програма потребно је позвати наредбу:

```
start rmiregistry
```

која покреће регистратор објеката.

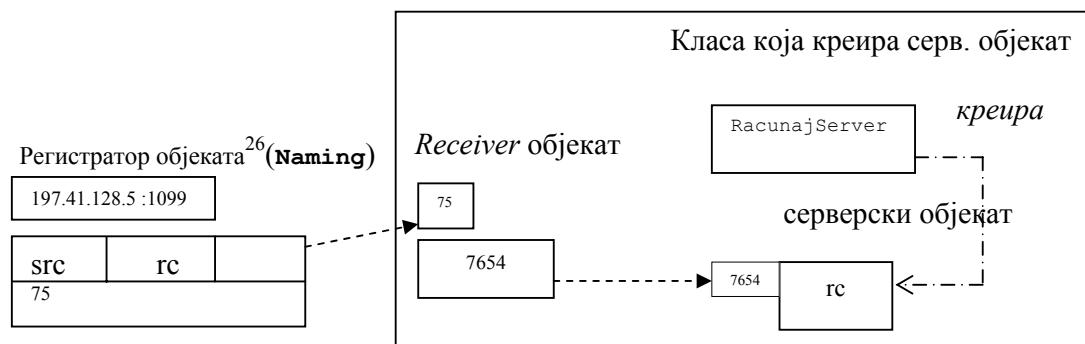
Након тога се позива наредба:

```
start java -Djava.rmi.server.codebase=http://127.0.0.1:9000/Download/
RacunajServer
```

која покреће серверски програм.

4.2.3.1 Регистровање серверских објеката

Серверски програм



²⁶ Sun-ов термин за регистратор објеката је **bootstrap registry service**. Регистратор објеката је реализован преко Naming класе која садржи један скуп static метода: lookup, bind, unbind, rebind и list које користе клијентски и серверски програм код RMI-а.

Процедура регистровања серверског објекта и његово повезивања са регистратором објектата:

- Прво се подиже регистратор објекта на порту 1099.
- Креира се серверски објекат на неком порту (7654)
- Серверски објекат се региструје код регистратора објекта. Стварни назив објекта (`rc`) се повезује са симболичким називом објекта (`src`). Регистратор добија референцу на `receiver` објекат који садржи референцу (порт) на серверски објекат.

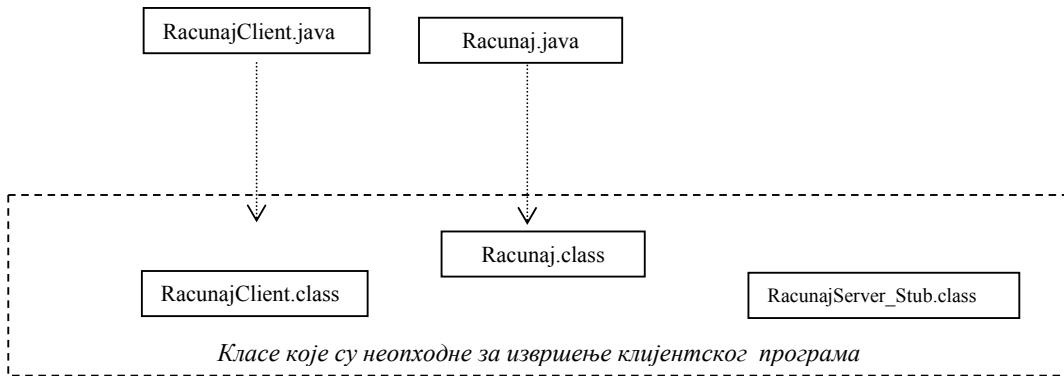
4.2.4 Структура клијентског програма

Клијентски програм у контексту RMI технологије садржи:

Интерфејс (нпр. `Racunaj`) који дефинише скуп операција (метода) које позива клијентски програм.

Класу (нпр. `RacunajClient`) која имплементира клијентски програм.

Када се клијентски програм компајлира добијају се две class датотеке (нпр. `Racunaj.class`, и `RacunajClient.class`). Поред наведене две класе потребно је на клијентској машини да се налази и стуб класа (нпр. `RacunajServer_Stub.class`) да би клијентски програм могао да се изврши.



Извршавање клијентског програма

Пре извршења клијентског програма потребно је на серверској машини подигнути: програм за регистровање серверских објеката и серверски програм.

Након тога се позива наредба:

```
java RacunajClient
```

која покреће клијентски програм.

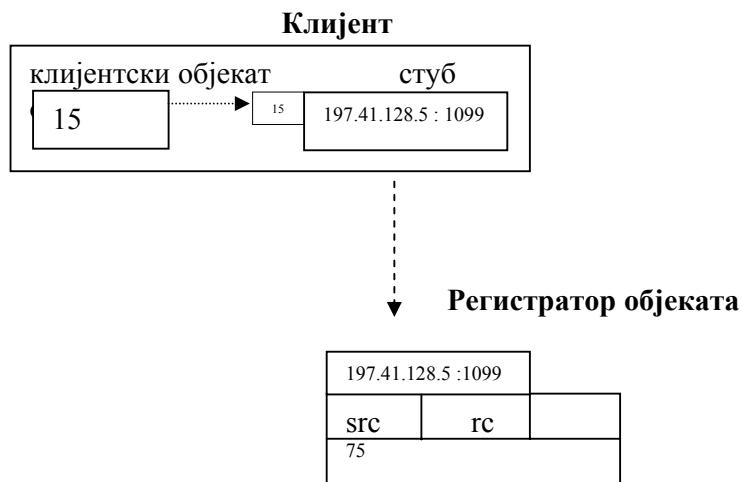
```
java -Djava.security.policy = client.policy RacunajClient
```

4.2.4.1 Повезивање клијента са стуб класом

Клијент се повезује са стуб класом на следећи начин:

- Клијент покушава да се повеже са регистратором објекта.
- Уколико је повезивање успешно извршено регистратор објекта шаље клијенту стуб класу.

- Стуб класа на почетку извршења клијентског програма није учитана у виртуалну машину²⁷. Виртуелна машина се пуни са стубом класом у време извршења програма када је регистратор објекта пошаље до клијента.



4.2.5 Повезивање клијентског са серверским објектом

Да би клијентски објекат могао да се повеже са серверским објектима потребно је да буду задовољени следећи предуслови:

- Треба омогућити клијенту могућност да приступи порту 1099 на серверској машини на коме је подигнут регистратор објекта.
- Треба омогућити клијенту могућност да приступи свим оним портовима (нпр. 7654) на којима су подигнути серверски објекти.
- Треба омогућити клијенту да приступи порту URL-а на коме се налази стуб класа.

Могућност повезивања клијента са серверским објектом се постиже помоћу *policy* датотеке²⁸ (нпр. *client.policy*) и менаџера заштите²⁹. *Policy* датотека има следећи садржај:

```
grant {permission java.net.SocketPermission "IP адреса сервера", "connect";};
```

За конкретан пример то изгледа овако:

```
grant {permission java.net.SocketPermission "197.41.128.5:1024-65535", "connect";};
```

Опсег портова се даје од 1024 до 65535 јер се не зна на ком ће се порту подигнути серверски објекти и где ће се налазити стуб датотека.

²⁷ Да би Јава програм могао да се изврши, Јавине *class* датотеке морају бити учитане у виртуелну машину. Учитавање ради специјалан део виртуелне машине који се зове пуњач класа (*class loader*).

²⁸ *Policy* датотеку чита и контролише менаџер заштите који мора бити подигнут на клијентском програму.

²⁹ *Policy* датотеку чита и контролише менаџер заштите и он омогућава клијентском програму да приступи или не приступи до серверских објеката.

4.2.6 Примери за RMI

4.2.6-1 Програмски захтев: Написати клијентски програм који позива серверски програм да му пошаље поздравну поруку.

Серверски програм:

```
/*
@author Sinisa Vlajic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
import java.rmi.*;
import java.rmi.server.*;

public class DobarDanServer {
    private int vrednost;

    public static void main(String[] args) {
        try {
            DobarDanImpl serverskiObjekat = new DobarDanImpl();
            Naming.rebind("SO", serverskiObjekat);
            System.out.println("Dobar dan - server je pokrenut!");
        } catch (Exception e) {
            System.err.println("Dobar dan greska: " + e);
        }
    }
}
```

```
import java.rmi.*;
public interface DobarDan extends Remote {
    public String vratiPozdrav() throws RemoteException;
}
```

```
import java.rmi.*;
import java.rmi.server.*;

public class DobarDanImpl extends UnicastRemoteObject implements DobarDan {
    private int vrednost;
    public DobarDanImpl() throws RemoteException {
        super();
        vrednost = 0;
    }
    public String vratiPozdrav() throws RemoteException {
        vrednost++;
        return "Dobar dan!!!. Broj poziva:" + vrednost;
    }
}
```

Поред наведених Јава програма направили смо и 2 bat датотеке ради покретање серверског програма:

```
javac DobarDanServer.java
rmic DobarDanImpl
copy DobarDan.class d:\Java\Download
copy DobarDanImpl_stub.class d:\Java\Download
cd rmiregister
start rmiregistry
```

```
start java -Djava.rmi.server.codebase=http://197.41.128.5:9000/Download/ DobarDanServer
```

Клијентски програм:

```
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
public class DobarDanKlijent {
    public static void main(String[] args) {
        System.setSecurityManager(new RMISecurityManager());
        try {
            String adresaServera = "197.41.128.5";
            // Dodeljivanje stub objekta do klijenta.
            DobarDan klijentskiObjekat = (DobarDan)
                Naming.lookup("rmi://" + adresaServera + "/SO");
            // Poziv metode udaljenog objekta.
            String poruka = klijentskiObjekat.vratiPozdrav();
            System.out.println(poruka);
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

```
import java.rmi.*;

public interface DobarDan extends Remote {
    public String vratiPozdrav() throws RemoteException;
}
```

```
grant {permission java.net.SocketPermission "197.41.128.5:1024-65535", "connect";};
```

Поред наведених Јава програма направили смо и *bat* датотеку ради покретања клијентског програма:

```
javac DobarDanKlijent.java
java -Djava.security.policy=client.policy DobarDanKlijent
pause
```

4.2.6-2 Програмски захтев: Написати серверски програм који сабира два броја која му шаље клијентски програм. Резултат (збир два броја) серверски програм враћа до клијенте.

Серверски програм:

```
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

import java.rmi.*;
import java.rmi.server.*;

public class RacunajServer {
    private int vrednost;
    public static void main(String[] args) {
        try {
            RacunajImpl serverskiObjekat = new RacunajImpl();
            Naming.rebind("SO", serverskiObjekat);
            System.out.println("Server sabiranja je pokrenut!");
        } catch (Exception e) {
            System.err.println("Server sabiranja ima gresku: " + e);
        }
    }
}
```

```
import java.rmi.*;
public interface Racunaj extends Remote {
    public double saberi(double broj1, double broj2) throws RemoteException;
}
```

```
import java.rmi.*;
import java.rmi.server.*;

public class RacunajImpl extends UnicastRemoteObject implements Racunaj {

    public RacunajImpl() throws RemoteException {
        // Ovaj konstruktor sa super() mora obavezno da se pokrene.
        super();
    }

    public double saberi(double broj1, double broj2) throws RemoteException {
        return broj1 + broj2;
    }
}
```

Поред наведених Јава програма направили смо и 2 *bat* датотеке ради покретање серверског програма:

```
javac RacunajServer.java
rmic RacunajImpl
copy Racunaj.class d:\Java\Download
copy RacunajImpl_stub.class d:\Java\Download
cd rmiregister
start rmiregistry
```

```
start java -Djava.rmi.server.codebase=http://197.41.128.5:9000/Download/ RacunajServer
```

Клијентски програм:

```
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;

public class RacunajKlijent {
    public static void main(String[] args) throws NumberFormatException {
        System.setSecurityManager(new RMISecurityManager());
        try {
            String adresaServera = "197.41.128.5";
            double broj1 = Double.parseDouble(args[0]);
            double broj2 = Double.parseDouble(args[1]);
            // Dodeljivanje stub objekta do klijenta.
            Racunaj klijentskiObjekat = (Racunaj) Naming.lookup("rmi://" + adresaServera + "/SO");
            // Poziv metode udaljenog objekta.
            double zbir = klijentskiObjekat.saberi(broj1, broj2);
            System.out.println("Zbir brojeva: " + broj1 + " i " + broj2 + " je: " + zbir );
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

```
import java.rmi.*;
public interface Racunaj extends Remote {
    public double saberi(double broj1, double broj2) throws RemoteException;
}
```

```
grant {permission java.net.SocketPermission "197.41.128.5:1024-65535", "connect"; };
```

Поред наведених Јава програма направили смо и *bat* датотеку ради покретање клијентског програма:

```
javac RacunajKlijent.java
java -Djava.security.policy=client.policy RacunajKlijent 6.78 4.56
pause
```

4.2.6-3 Програмски захтев: Написати серверски програм који пуни објекат студент са вредностима које саље клијентски програм. Омогућити да серверски програм може да врати студент објекат до клијентског програма.

Серверски програм:

```
/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.rmi.*;
import java.rmi.server.*;

public class NapuniServer {
    public static void main(String[] args) {
        try {
            NapuniImpl serverskiObjekat = new NapuniImpl();
            Naming.rebind("SO", serverskiObjekat);
            System.out.println("Napuni - server je pokrenut!");
        } catch (Exception e) {
            System.err.println("Server ima gresku: " + e);
        }
    }
}
```

```
import java.rmi.*;

public interface Napuni extends Remote {
    public void ubaci(String brojIndeksa, String Ime, int brojSemestra) throws RemoteException;
    public Student citaj() throws RemoteException;
}
```

```
import java.rmi.*;
import java.rmi.server.*;
public class NapuniImpl extends UnicastRemoteObject implements Napuni {
    private String brojIndeksa;
    private String Ime;
    private int brojSemestra;

    public NapuniImpl() throws RemoteException {
        super();
    }

    public void ubaci(String brojIndeksal, String Imel, int brojSemestral) throws RemoteException {
        brojIndeksa = new String(brojIndeksal);
        Ime = new String(Imel);
        brojSemestra = brojSemestral;
        System.out.println("Napunjen je student:" + brojIndeksa + " cije je ime: " + Ime + ". Broj semestra: " + brojSemestra);
    }

    public Student citaj() throws RemoteException {
        Student st = new Student();
        st.brojIndeksa = new String(brojIndeksa);
        st.Ime = new String(Ime);
        st.brojSemestra = brojSemestra;
        return st;
    }
}
```

```

import java.io.*;

public class Student implements Serializable {
    public String brojIndeksa;
    public String Ime;
    public int brojSemestra;
    public Student(){brojIndeksa =""; Ime="";brojSemestra=0;}
}

```

Поред наведених Јава програма направили смо и 2 *bat* датотеке ради покретање серверског програма:

```

javac NapuniServer.java
rmic NapuniImpl
copy Napuni.class d:\Java\Download
copy NapuniImpl_stub.class d:\Java\Download
copy Student.class d:\Java\Download
cd rmiregister
start rmiregistry

```

```
start java -Djava.rmi.server.codebase=http://197.41.128.5:9000/Download/ NapuniServer
```

Клијентски програм:

```

/*
@auther Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;

public class NapuniKlijent {
    public static void main(String[] args) {
        System.setSecurityManager(new RMISecurityManager());
        try {
            String adresaServera = "197.41.128.5";
            String brojIndeksa = new String("123-04");
            String Ime = new String("Pera Peric");
            int brojSemestra=2;
            Student st = new Student();
            // Dodeljivanje stub objekta do klijenta.
            Napuni KlijentskiObjekat = (Napuni) Naming.lookup("rmi://" + adresaServera + "/SO");
            // Poziv metode udaljenog objekta.
            klijentskiObjekat.ubaci(brojIndeksa,Ime,brojSemestra);
            st=klijentskiObjekat.citaj();
            System.out.println("Procitan je student:" + st.brojIndeksa +
                " cije je ime: " + st.Ime + ". Broj semestra: " + st.brojSemestra);
        }catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}

```

```

import java.rmi.*;

public interface Napuni extends Remote {
    public void ubaci(String brojIndeksa, String Ime, int brojSemestra) throws RemoteException;
    public Student citaj() throws RemoteException;
}

```

```

import java.io.*;

public class Student implements Serializable {
    public String brojIndeksa;
    public String Ime;
    public int brojSemestra;
    public Student(){brojIndeksa =""; Ime="";brojSemestra=0;}
}
grant {permission java.net.SocketPermission "197.41.128.5:1024-65535", "connect"; };

```

Поред наведених Јава програма направили смо и *bat* датотеку ради покретање клијентског програма:

```
K5. PokreniKlijenta.bat
javac NapuniKlijent.java
java -Djava.security.policy=client.policy NapuniKlijent
pause
```

Задаци

31РМИ: Програмски захтев: Написати клијентски програм који приhvата `String` као аргумент који затим шаље до серверског програма. Серверски програм одређује број самогласника прихваћеног стрингу и тај број враћа назад до клијентског програма.

32РМИ: Програмски захтев: Написати клијентски програм који шаље низ бројева до серверског програма. Серверски програм одредује позитивне елементе из низа и враћа их назад до клијентског програма.

33РМИ: Програмски захтев: Написати клијентски програм који креира и пуни објекат а затим га шаље до серверског програма. Серверски програм контролише вредности атрибута и враћа обрађени објекат са исправним вредностима атрибута до клијентског програма.

4.2.7 Генерички RMI пример

Када се поставља у генералном смислу комуникација између клијентског и серверског програма помоћу RMI тада се раде следеће активности:

1. Одређује се име интерфејса, јер од њега зависе имена већине осталих класа у програму. Нпр. Ако интерфејс назовемо **Inter** тада ће **серверски програм** имати следећи садржај:

```
import java.rmi.*;
// U interfejsu se navodi jedna ili više udaljenih metoda koje treba da se implementiraju.
public interface Inter extends Remote{
    public double m1(...) throws RemoteException;
    public double m2(...) throws RemoteException;
    ...
    public double mn(...) throws RemoteException;
}
```

```
import java.rmi.*;
import java.rmi.server.*;

public class InterImpl extends UnicastRemoteObject implements Inter{
    public Inter() throws RemoteException {
        // Ovaj konstruktor sa super() mora obavezno da se pokrene.
        super();
    }
    // Implementacija udaljenih metoda
    public double m1(...) throws RemoteException {...}
    public double m2(...) throws RemoteException {...}
    ...
    public double mn(...) throws RemoteException {...}
}
```

```

import java.rmi.*;
import java.rmi.server.*;
public class InterServer{
    private int vrednost;
    public static void main(String[] args){
        try {
            InterImpl serverskiObjekat = new InterImpl();
            Naming.rebind("SO",serverskiObjekat);
            System.out.println("Server je pokrenut!");
        } catch (Exception e) {
            System.err.println("Server ima gresku: " + e);
        }
    }
}

```

```

// Ukoliko objekat treba da se vrati kroz mrežu on treba da se serijalizuje.
import java.io.*;
public class Entitet1 implements Serializable{
    ...
}
public class Entitet2 implements Serializable{
    ...
}
public class Entitetm implements Serializable{
    ...
}

```

Поред наведених Јава програма направили смо и 2 *bat* датотеке ради покретање серверског програма:

```

javac InterServer.java
rmic InterImpl
copy Inter.class d:\Java\Download
copy InterImpl_stub.class d:\Java\Download
cd rmiregister
start rmiregistry

```

```

start java -Djava.rmi.server.codebase= URL serverske mašine gde se nalazi folder za download/
InterServer

```

Клијентски програм:

```

/*
@author Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
public class RacunajKlijent{
    public static void main(String[] args) throws NumberFormatException{
        System.setSecurityManager(new RMISecurityManager());
        try {
            String adresaServera =URL adresa servera;
            double broj1 = Double.parseDouble(args[0]);
            double broj2 = Double.parseDouble(args[1]);
            // Dodeljivanje stub objekta do klijenta.
            Racunaj klijentskiObjekat = (Racunaj) Naming.lookup("rmi://" +adresaServera+ "/SO");
            // Poziv metode udaljenog objekta.
            double zbir = klijentskiObjekat.saberi(broj1,broj2);
            System.out.println("Zbir brojeva: " + broj1 + " i" + broj2 + " je: " + zbir );
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}

```

```
import java.rmi.*;
// U interfejsu se navodi jedna ili više udaljenih metoda koje treba da se implementiraju.
public interface Inter extends Remote{
    public double m1(...) throws RemoteException;
    public double m2(...) throws RemoteException;
    ...
    public double mn(...) throws RemoteException;
}
```

```
import java.io.*;

public class Entitet1 implements Serializable{ ... }

public class Entitet2 implements Serializable{ ... }

public class Entitetm implements Serializable{
    ...
}
```

```
grant { permission java.net.SocketPermission,URL adresa serverske mašine sa opsegom portova
na kojima su podignuti objekti i okruženje serverskog programa , "connect"; };
```

Поред наведених Јава програма направили смо и *bat* датотеку ради покретање клијентског програма:

```
javac InterKlijent.java
java -Djava.security.policy=client.policy InterKlijent
pause
```

Напомена: Све оно што је у генеричком програму болдовано за конкретни пример треба да се специјализује.

5. ЗАШТИТА

Механизми заштите код Јаве су саставни део Јава технологије. Постоје три механизма која обезбеђују сигурност:

- Особине самог језика (контрола границе низа, конверзија типа, непостојање поинтерске аритметике,...).
- Контрола приступа програмском коду (приступ датотекама, приступ програму кроз мрежу,...).
- Означавање кода (*Code signing*), где аутори могу да користе стандардан криптографски алгоритам да ауторизују (аутентификују) Јава програмски код. Корисник кода може да одреди тачно ко је креирао код и ко га може мењати пошто је он означен.

Када се `class` датотеке напуне у виртуелну машину, контролише се њихов интегритет. У Јави је могуће направити сопствени пуњач класа (*class loader*). Пуњач класа ради са `SecurityManager` класом која контролише акције које програмски код може да изврши.

5.1 Пуњачи класа

Компајлер Јава програмског језика конвертује изворне наредбе Јаве у машинске наредбе хипотетичке машине, која се назива виртуална машина. Виртуална машина интерпретира машинске наредбе тако што их преслика у позиве машинских наредби конкретног оперативног система на коме се она извршава. На тај начин су индиректно повезане изворне Јаве наредбе са наредбама конкретног оперативног система на коме се извршава Јава програм односно његова виртуална машина. Наредбе које интерпретира виртуална машина су сачуване у датотеци са `class` екstenзијом.

Виртуелна машина интерпретира све оне класе које су потребне за извршење Јава програма.

Наводимо кораке која извршава виртуална машина код извешења следећег Јава програма:

```
import java.io.*;
import java.util.*;

class X extends Y
{ Z z;
  public static void main(String [] args) {F f;...m1(new D); ... m2();...}
  void m1(D d) {...}
  void m2() {E e; ...}
}
```

Прво се пуни `X.class` датотека.

Затим се пуне датотеке које садрже класе које су повезане са класом `X` `is-a` (`Y`) и `has` (`Z`) везом. То су `Y.class` и `Z.class` датотеке.

Виртуална машина тада извршава `main` методу.

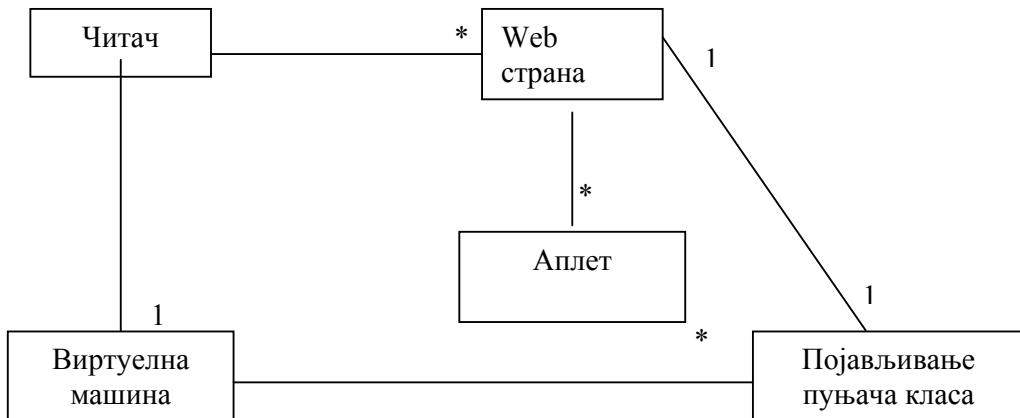
Ако `main` метода садржи допунске класе (`F, D`) или методе које она позива садрже допунске класе (`E`), датотеке у којима се оне налазе бивају напуњене. То су `F.class`, `D.class` и `E.class`.

Поред наведених класа виртуелна машина мора да напуни и системске класе које су потребне за извршење програма (`java.util.*` и `java.io.*`).

Корисник може да утиче на процес пуњења виртуелне машине са апликационим класама (`X, Y, ..., E`). Он неможе да утиче на процес пуњења системских класа.

Пуњач класа аплета (Applet class loader)

Читач (*browser*) за сваку *Web* страну покреће по једно појављивање пуњача класа аплета (ако *Web* страна има бар један аплет), који пуни виртуелну машину са аплетима који се налазе на тој *Web* страни. То значи да сваки пар *Web* страна-аплет има своје стање које је независно у односу на било који други пар *Web* страна-аплет. Исти аплети покренути на две различите *Web* стране се посматрају као два независна појављивања (објекта).



5.1.1 Писање сопственог пуњача класа

Када се пише сопствени пуњач класа потребно је да се имплементира апстрактна класа `ClassLoader`. Метода `loadClass()` у тој класи одређује како ће се напунити класа.

Код имплементације наведене методе потребно је:

- Проверити да ли је пуњач класа већ напунио ову класу. То значи да сопствени пуњач класа мора да чува називе свих оних класа које је предходно напунио. Имена класа он чува у некој колекцији (нпр. `hash` табела).
- Ако је то нова класа треба проверити да ли је то системска класа. Ако је то системска класа прекида се пуњење класе.
- Нова класа се пуни са низом бајтова (*bytecode*) из неке датотеке. Та датотека је узвари `class` датотека иако због криптоирања података она може бити сачувана у датотеци са неком другом екstenзијом.
- Позива се `defineClass` метода од `ClassLoader` класе да пренесе *bytecode* у виртуелну машину.
- Име нове класе пуњач класа памти у одговарајућој колекцији.
- Контролише се да ли су све класе које су потребне за извешење нове класе учитане у виртуелну машину како би нова класа могла да се изврши. Докле год је атрибут `resolve` постављен на `true` нова класа неће моћи да се изврши.

```

public class PunjacKlasa extends ClassLoader {

    ...

    // Kolekcija koja čuva imena klase koje je učitao u virtuelnu memoriju punjač klasa.
    private Map klase = new HashMap();
    ...

    protected synchronized Class loadClass(String ime, boolean resolve) throws
ClassNotFoundException
        // 1. Provera da li je punjač klasa već napunio ovu klasu.
    {
        Class klasa = (Class) klase.get(ime);

        // 2. Ako je nova klasa treba proveriti da li je to sistemska klasa.
        if (klasa == null) {
            try // Ako je sistemska klasa prekida se punjenje.
            {
                return findSystemClass(ime);
            } catch (ClassNotFoundException e) {
                System.out.println("Izuzetak poruka21:");
            } catch (NoClassDefFoundError e) {
                System.out.println("Izuzetak poruka22:");
            }
        }

        // 3. Nova klasa se puni sa nizom bajtova (bytecode) iz neke datoteke
        byte[] klasaBajtova = napuniKlasuBajtova(ime); //loadClassBytes
        if (klasaBajtova == null) {
            throw new ClassNotFoundException(ime);
        }

        // 4. Prenosi se bytecode nove klase u virtuelnu mašinu.
        klasa = defineClass(ime, klasaBajtova, 0, klasaBajtova.length);
        if (klasa == null) {
            throw new ClassNotFoundException(ime);
        }

        // 5. Ime nove klase pamti se u heš tabeli.
        klase.put(ime, klasa);
    }

    /* 6. Kontroliše se da li su sve klase koje su potrebne za izvršenje nove
     * klase učitane u virtuelnu mašinu
     * kako bi nova klasa mogla da se izvrši.
    */

    if (resolve) {
        resolveClass(klasa);
    }

    return klasa;
}
...
}

```

5.1.2 Енкриптоњање класе

Уколико постоји потреба да се `class` датотеке енкриптују (шифрирају, *encrypt*), како нико не би могао да им приступи осим онога који има програм за декрипцију (десифрирање, *decrypt*).

Једноставан начин криптоњања `class` датотеке је да сваки бајт датотеке повећамо или смањимо за неку вредност. У случају декрипције је потребно да се уради инверзна операција од операције која је коришћена код енкрипције. На пример уколико је код енкриптоња сваки бајт повећан за вредност 33, код декрипције он ће бити умањен за ту вредност. Ово је једноставан алгоритам који се може релативно лако пробити али он у суштини представља основу за алгоритме декриптоња који мењају вредности сваког бајта `class` датотеке по неком сложеном сценарију рачунања.

Наводимо пример програма помоћу кога се врши једноставно криптоњање програма:

```

/*
@author Sinisa Vlajic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

import java.io.*;

// args[0] - class datoteka koja se cita.
// args[1] - nova enkriptovana datoteka sa ekstenzijom enk.
// args[2] - kljuc

public class Kriptovanje {

    public static void main(String[] args) {
        args = new String[3];
        args[0] = "Mrezal.class";
        args[1] = "Mrezal.kri";
        args[2] = "33";

        try {
            FileInputStream in = new FileInputStream(args[0]);
            FileOutputStream out = new FileOutputStream(args[1]);
            int kljuc = Integer.parseInt(args[2]);
            int znak;
            while ((znak = in.read()) != -1) {
                // svaki bajt se umanjuje za vrednost kljuc promenljive, koja je u ovom
                // slučaju 33.
                byte b = (byte) (znak - kljuc); // enkriptovanje bajtova
                System.out.print((byte) znak);
                out.write(b);
            }
            in.close();
            out.close();
        } catch (IOException e) {
            System.out.println("Greska:" + e);
        }
    }
}

```

5.1.3 Пример пуњача класе који памти енкриптовану класу

Уколико поћемо од претпоставке да је class датотека Mrezal.class енкриптована, преко програма који се налази у класи Kriptovanje, у датотеку са екстензијом Mrezal.enk тада програм за пуњење енкриптоване класе у виртуелну машину има следећи изглед:

```

/*
@author Sinisa Vlajic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
// Primer PK1
import java.util.*; // klasa HashMap
import java.lang.reflect.*; // klasa Method
import java.io.*;

public class PunjacKlasa extends ClassLoader {

    // Vrednost na koju je enkriptovana class datoteka Mrezal.class.
    private int kljuc;
    // Kolekcija koja čuva imena klase koje je učitao u virtuelnu memoriju punjač klasa.
    private Map klase = new HashMap();

    public PunjacKlasa(int kljuc1) {
        kljuc = kljuc1;
    }
    public static void main(String[] args) {
        // Kreira se punjač klasa i inicijalizuje se na ključ koji ima vrednost 33.
        PunjacKlasa pk = new PunjacKlasa(33);
        pk.izvrsiKlasu("Mrezal");
    }
}

```

```

public void izvrsiKlasu(String ime) {
    try {
        // Kreiranje punjaca klase.
        ClassLoader punjac = new PunjacKlasa(kljuc);
        // Punjač puni virtuelnu memoriju sa klasom Mrezal.class.
        Class klasa = punjac.loadClass(ime);
        String[] args = new String[] {};
        // Kreiranje objekta za poziv main metode klase Mrezal.class.
        Method m = klasa.getMethod("main", new Class[]{args.getClass()});
        // Poziv main metode klase Mrezal.class.
        m.invoke(null, new Object[]{args});
    } catch (Throwable e) {
        System.out.println("Izuzetak: " + e);
    }
}
protected synchronized Class loadClass (String ime, boolean resolve) throws
ClassNotFoundException {
    // Provera da li je punjač klasa već napunio ovu klasu.
    Class klasa = (Class) klase.get(ime);
    // Ako je nova klasa treba proveriti da li je to sistemska klasa.
    if (klasa == null) {
        try // Ako je sistemska klasa prekida se punjenje.
        {
            return findSystemClass(ime);
        } catch (ClassNotFoundException e) {
            System.out.println("Izuzetak poruka21:");
        } catch (NoClassDefFoundError e) {
            System.out.println("Izuzetak poruka22:");
        }
        // Nova klasa se puni sa nizom bajtova (bytecode) iz neke datoteke
        byte[] klasaBajtova = napuniKlasuBajtova(ime); //loadClassBytes
        if (klasaBajtova == null) {
            throw new ClassNotFoundException(ime);
        }

        // Prenosi se bytecode nove klase u virtuelnu mašinu.
        klasa = defineClass(ime, klasaBajtova, 0, klasaBajtova.length);
        if (klasa == null) {
            throw new ClassNotFoundException(ime);
        }
        // Ime nove klase pamti se u heš tabeli.
        klase.put(ime, klasa);
    }
    // Kontroliše se da li su sve klase koje su potrebne za izvršenje nove
    // klase učitane u virtuelnu mašinu
    // kako bi nova klasa mogla da se izvrši.
    if (resolve) {
        resolveClass(klasa);
    }
    return klasa;
}
private byte[] napuniKlasuBajtova(String ime) {
    String imen = ime + ".enk";
    FileInputStream in = null;
    try {
        in = new FileInputStream(imen);
        ByteArrayOutputStream bafer = new ByteArrayOutputStream();
        int znak;
        while ((znak = in.read()) != -1) {
            // Svaki bajt se povećava za vrednost ključ promenljive koja je 33.
            // Ovde se radi inverzna operacija onoj operaciji koja je korišćena kod
            // enkriptovanja bajtova.
            byte b = (byte) (znak + kljuc); // dekriptovanje bajtova
            System.out.print(b);
            bafer.write(b);
        }
        in.close();
        return bafer.toByteArray();
    } catch (IOException e) {
        if (in != null) {
            try {
                in.close();
            } catch (IOException e1) {
            }
        }
        return null;
    }
}

```

Напомена: Ако има више бајт код класа које треба пренети у виртуелну машину тада све оне класе које су енкриптоване треба напунити преко сопственог пуњача класа. Остале класе није потребно експлицитно пунити, оне ће аутоматски бити напуњене преко методе `resolveClass`.

Задатак ЗПК1: Направити програм за енкриптоање бајтова class датотеке.

Задатак ЗПК2: Направити сопствени пуњач класе који ће да напуни класу која је криптована преко програма који је урађен у задатку ЗПК1.

5.2 Byte code верификација

Када пуњач класа пуни *bytecode* у виртуелну машину тада се дешава контрола пуњења преко **верификатора** (*verifier*) који контролише наредбе и онемогућава им да се изврше уколико праве неки проблем. Све класе, изузев системских класа се верификују. Могуће је деактивирати верификацију са наредбом:

```
java -noverify Mreza1
```

Деактивирање верификације се не препоручује јер може довести до озбиљних проблема у извршењу програма.

Верификатор чини следеће акције код контроле наредби:

- Променљиве морају бити иницијализоване пре него што се користе.
- Аргументи (стварни параметри) и параметри (формални параметри) метода се морају слагати по типу код позива метода.
- Начин приступа до чланица класа мора бити задовољен.
- Локалним променљивима се не може приступити изван метода у којој су декларисане.
- *Runtime* стак за чување параметара и локалних променљивих не сме да се прекорачи (*overflow*).

Ако један од наведених услова није задовољен класа неће моћи да се напуни.

5.2.1 Промена class датотека преко hex едитора

Међутим ако је неко упознат са програмирањем у асемблеру и ако зна да ради са *hex* едитором он може заменити садржај class датотеке тако да се она извршава на другачији начин у односу на њено првобитно понашање.

На пример:

```
// Primer BKV1
class Verifikacija{
    public static void main(String[] args){
        int x = 3;
        int y = 1;
        int r = x + y;
        r = r + 3;
    }
}
```

Уколико се погледа мнемонични облик class датотеке са:

```
javap -c Verifikacija
```

добиће се:

```
Compiled from "Verifikacija.java"
```

```

class Verifikacija extends java.lang.Object{

Verifikacija();
Code:
  0:  aload_0
  1:  invokespecial #1; //Method java/lang/Object."<init>":()V
  4:  return

public static void main(java.lang.String[]);
Code:
  0:  iconst_3
  1:  istore_1
  2:  iconst_1
  3:  istore_2
  4:  iload_1
  5:  iload_2
  6:  iadd
  7:  istore_3
  8:  iload_3
  9:  iconst_3
 10:  iadd
 11:  istore_3
 12:  return

}

```

Наредбе извornог програма се пресликају у мнемонички облик на следећи начин:

a) int x = 3;

преслика се у

0: iconst_3 (Константа која има вредност 3,
1: istore_1 додељује се променљивој 1. Променљива 1 је уствари x)

b) int y = 1;

преслика се у

2: iconst_1 (Константа која има вредност 1,
3: istore_2 додељује се променљивој 2. Променљива 2 је уствари y)

c) int r = x + y;

преслика се у

4: iload_1 (Променљива 1 памти своју вредност у меморији,
5: iload_2 променљива 2 памти своју вредност у меморији,
6: iadd запамћене вредности се сабирају и
7: istore_3 додељују се до променљиве 3. Променљива 3 је уствари r.)

d) r = r + 3;

преслика се у

8: iload_3 (Променљива 3 памти своју вредност у меморији i
9: iconst_3 константа која има вредност 3 се
10: iadd сабирају и
11: istore_3 додељују се до променљиве 3.)

Свака од наредби које су у мнемоничком облику је повезана са њеном хексадецималном вредношћу:

0: iconst_3	06
1: istore_1	3C
2: iconst_1	04
3: istore_2	3D
4: iload_1	1B
5: iload_2	1C
6: iadd	60
7: istore_3	3E

```

8: iload_3           1D
9: iconst_3          06
10: iadd             60
11: istore_3         3E
12: return            B1

```

Из наведеног може да се види да хексадецималне вредности инструкција једнозначно одређују мнемоничке наредбе:

одређује `iconst_3`

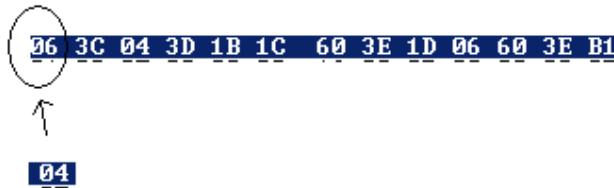
одређује `iadd`

`3E` одређује `istore_3`

Инструкције у хексадецималном облику се могу видети у hex едитору:

0:	CA	FE	BA	BE	00	00	00	2E	00	0F	0A	00	03	00	0C	07	11	1F	..
10:	00	0D	07	00	0E	01	00	06	3C	69	6E	69	74	3E	01	00	<init>
20:	03	28	29	56	01	00	04	43	6F	64	65	01	00	0F	4C	69	Code...Li
30:	6E	65	4E	75	6D	62	65	72	54	61	62	6C	65	01	00	04	neNumberTable...
40:	6D	61	69	6E	01	00	16	28	5B	4C	6A	61	76	61	2F	6C	main...<[Ljava/l
50:	61	6E	67	2F	53	74	72	69	6E	67	3B	29	56	01	00	0A	ang/String;>U...
60:	53	6F	75	72	63	65	46	69	6C	65	01	00	11	56	65	72	SourceFile...Ver
70:	69	66	69	6B	61	63	69	6A	61	2E	6A	61	76	61	0C	00	if ikacija.java...
80:	04	00	05	01	00	0C	56	65	72	69	66	69	6B	61	63	69	Verifikaci...
90:	6A	61	01	00	10	6A	61	76	61	2F	6C	61	6E	67	2F	4F	ja...java/lang/0
A0:	62	6A	65	63	74	00	20	00	02	00	03	00	00	00	00	00	bject...
B0:	02	00	00	00	04	00	05	00	01	00	06	00	00	00	1D	00
C0:	01	00	01	00	00	00	05	2A	B7	00	01	B1	00	00	00	01	*п..
D0:	00	07	00	00	00	06	00	01	00	00	00	01	00	09	00	08
E0:	00	09	00	01	00	06	00	00	00	35	00	02	00	04	00	00	5
F0:	00	0D	06	3C	04	3D	1B	1C	60	3E	1D	06	60	3E	B1	00	<.=.:>:>'
100:	00	00	01	00	07	00	00	00	16	00	05	00	00	00	05	00
110:	02	00	06	00	04	00	07	00	08	00	08	00	0C	00	09	00
120:	01	00	0A	00	00	00	02	00	0B

Уколико се нпр. у hex editoru уместо `06` стави `04`,



добиће се ефекат као да смо изменили програмски код.

```

class Verifikacija{
    public static void main(String[] args){
        int x = 1;
        int y = 1;
        int r = x + y;
        r = r + 3;
    }
}

```

У суштини ми нисмо мењали изворни код већ смо променили class датотеку. На основу наведеног може се закључити да је понашање програма Verifikacija другачије у односу на њено првобитно понашање.

Задатак ЗБКВ1:

Промени наредбе из хексадецималног облика програма **БКВ1**:

06 3C 04 3D 1B 1C 60 3E 1D 06 60 3E B1

тако да се добије ефекат:

```
class Verifikacija{
    public static void main(String[] args){
        int x = 3;
        int y = 1;
        int r = x + y;
        r = r + 1;
    }
}
```

Задатак БКВ2:

Промени наредбе из хексадецималног облика програма БКВ1:

06 3C 04 3D 1B 1C 60 3E 1D 06 60 3E B1

тако да се добије ефекат:

```
class Verifikacija{
    public static void main(String[] args){
        int x = 3;
        int y = 3;
        int r = x + y;
        r = r + 3;
    }
}
```

5.2.2 Рекомпајлери class датотека

Поред наведеног начина промене класс датотеке преко *hex* један једноставнији начин, коришћењем декомпајлера. Они су у стању да *class* датотеку пребаце у изворни облик. На тај начин је пружена могућност да се мења изворни програм који након тога може да се компајлира.

5.3 Менаџери заштите (Security managers) и дозволе (permissions)

Након што је класа напуњена у виртуелну машину помоћу пуњача класа и тестирана помоћу верификатора она се даље контролише помоћу **менаџера заштите**. Менаџер заштите је класа која контролише да ли операције програма могу да се изврше (да ли су дозвољене). Он контролише следеће операције:

- Креирање новог пуњача класа од текуће нити.
- Креирање подпроцеса од текуће нити.
- Заустављање виртуелне машине од текуће нити.
- Приступ специфичном пакету од текуће нити.
- ...
- Брисање специфичне датотеке од текуће нити.
- Читање или уписивање у датотеке од текуће нити.
- Отварање сокет конекције на специфичном хосту и броју порта.
- ...
- Приступ једне класе члановима друге класе.

Подразумевано, код извршења Јава апликације, није укјучен менаџер заштите, тако да су све операције дозвољене. Са друге стране када се покреће аплет аутоматски је инсталiran менаџер заштите који је прилично рестриктиван. У једном програму може бити инсталiran највише један менаџер заштите.

Наједноставнији начин постављање дозвола, без промене постојећег програмског кода, је помоћу политику датотеке. Уколико желимо да дозволимо да се чита и уписује у датотеку *Mrezal.enk* тада политику датотека, којој смо дали име *policy.dat*, има следећи садржај:

```
grant {
    permission java.io.FilePermission "Mrezal.enk", "read,write";
};
```

У случају програма ЕНК1,

```

/*
@author Sinisa Vlajic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
import java.io.*;
...
public class Kriptovanje {

    public static void main(String[] args) {
        System.setSecurityManager(new SecurityManager());
        args = new String[3];
        args[0] = "Mrezal.class";
        args[1] = "Mrezal.kri";
        args[2] = "33";
        try {
            FileInputStream in = new FileInputStream(args[0]);
            FileOutputStream out = new FileOutputStream(args[1]);
            int kljuc = Integer.parseInt(args[2]);
            int znak;
            while ((znak = in.read()) != -1) {
                byte b = (byte) (znak - kljuc); // enkriptovanje bajtova
                System.out.print((byte) znak);
                out.write(b);
            }
            in.close();
            out.close();
        } catch (IOException e) {
            System.out.println("Greska:" + e);
        }
    }
}

```

уноси се допунска наредба `System.setSecurityManager(new SecurityManager())` ради креирања менаџера заштите.

Уколико желимо да наведени програм повежемо са *policy* датотеком извршићемо наредбу:

```
java -Djava.security.policy=polocy.dat Kriptovanje
```

Уколико би позвали наредбу:

```
java Kriptovanje
```

јавила би се следећа порука:

```

java.security.AccessControlException: access denied (java.io.FilePermission Mrezal.enk write)
    at java.security.AccessControlContext.checkPermission(Unknown Source)
    at java.security.AccessController.checkPermission(Unknown Source)
    at java.lang.SecurityManager.checkPermission(Unknown Source)
    at java.lang.SecurityManager.checkWrite(Unknown Source)
    at java.io.FileOutputStream.<init>(Unknown Source)
    at java.io.FileOutputStream.<init>(Unknown Source)
    at Kriptovanje.main(Kriptovanje.java:18)
Exception in thread "main"

```

Задатак М31: Креирати произвољну датотеку. Напунити је са произвољним садржајем. Омогућити да она може да се чита. Онемогућити да се може нешто уписивати.

5.4 Дигитални потпис

Да би се схватио дигитални потпис потребно је да се схвати концепт криптографског јавног кључа. Он је заснован на појмовима приватни и јавни кључ. Идеја је у томе да свако зна ваш јавни кључ а да само ви знате ваш приватни кључ. Приватни и јавни кључ су повезани неком математичком релацијом коју је тешко, готово немогуће открити³⁰.

Процедура која се користи код креирања и верификација дигиталног потписа је следећа:

- Генерише се приватни и јавни кључ
- Ко шаље поруку креира потпис и повезује га са приватним кључем.
- Јавни кључ се прослеђује до примаоце поруке.
- Потпис се повезује са самом поруком која ће бити послата.
- Потпис се означава(*signed signature*).
- Потпис и порука се шаљу до примаоца поруке.
- Примаоц верификује потпис и поруку на основу јавног кључа.

Постоје различити алгоритми за генерирање кључева и креирање потписа као што су *DSA* алгоритам (*Digital Signature Algorithm*) и *RSA* алгоритам (алгоритам енкрипције који су направили *Rivest, Shamir и Adleman*).

Јавино окружење (пакет заштите) садржи *DSA*³¹.

Пример ДП1:

```
/*
@author Sinisa Vlajic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

import java.security.*;

public class DigitalniPotpis {
    public static void main(String[] args) {
        try {
            KeyPairGenerator genKljуча = KeyPairGenerator.getInstance("DSA");
            SecureRandom sr = new SecureRandom();
            // kljucevi ce imati duzinu 512 bitova (64 bajta).
            genKljуча.initialize(512, sr);
            // Generise se privatni i javni ključ
            KeyPair parKljuceva = genKljуча.generateKeyPair();
            PublicKey javniKljuc = parKljuceva.getPublic();
            PrivateKey privatniKljuc = parKljuceva.getPrivate();

            // Kreira se potpis preko DSA algoritma.
            Signature potpisAlg = Signature.getInstance("DSA");
            // Povezivanje potpisa sa privatnim kljucem.
            potpisAlg.initSign(privatniKljuc);

            String poruka = "Danas je lep dan";
            // Potpis se povezuje sa porukom koja ce biti poslata.
            potpisAlg.update(poruka.getBytes());
            // Potpis se oznacava.
            byte[] potpis = potpisAlg.sign();

            // Kreira se verifikator preko DSA algoritma.
            Signature verifikatorAlg = Signature.getInstance("DSA");
            // Povezuje se javni kljuc sa verifikatorom.
            verifikatorAlg.initVerify(javniKljuc);
            // Poruka se dodeljuje verifikatoru.
            verifikatorAlg.update(poruka.getBytes());
        }
    }
}
```

³⁰ То практично значи да неко треба да нађе математичку релацију између јавног кључа и приватног кључа који је скривен (разбациан) у некој датотеци, при чему је та датотека испуњена и са неким другим садржајима а не само приватним кључем.

³¹ RSA алат се може купити преко сајта www.rsa.com.

```

    // Verifikator proverava da li je potpis u redu.
    if (!verifikatorAlg.verify(potpis)) {
        System.out.println("Potpis nije odgovarajuci!");
    } else {
        System.out.println("Potpis je odgovarajuci!");
    }

} catch (Exception e) {
    System.out.println("Izuzetak!" + e);
}
}
}

```

Задатак ЗДП1: Направити клијентски програм који ће генерисати клучеве преко *ДСА* алгоритма. Послати јавни кључ до серверског програма. Направити поруку и потпис на клијентској страни и послати их на верификацију серверском програму. Серверски програм треба да јави клијентском програму да ли је верификација успешна.

5.5 Аутентификација

Када примате поруку са потписом од некога може се јавити проблем аутентификације (авторизације) односно идентификовања онога ко шаље поруку. Постоје два начина аутентификације:

Преко посредника кога познају и пошиљалац и прималац поруке. Посредник ће дати јавни кључ премајуци поруке.

Преко потписа кога прави посредник у комуникацији између пошиљаоца и премаоца поруке. Посредник прави свој приватни кључ и повезује га са јавним кључем пошиљаоца. Прималац добија јавни кључ посредника, сазнаје његов приватни кључ а преко њега и јавни кључ пошиљаоца.

5.6 Означавање аплета

Аплети не могу да приступе до система датотека текуће машине на којој се налази читач (*browser*) који је скинуо аплет. У неким случајевима је потребно да се омогући аплетима да приступе до система датотеке текуће машине. То је могуће урадити на следећи начин:

Комарајирају се све оне класе (x_1, x_2, \dots, x_n) које су потребне за извршење аплета:

```

javac X1.java
javac X2.java

javac Xn.java

```

Прави се *jар* датотека у коју се убацују *class* датотеке *X1.class* *X2.class...,Xn.class*:

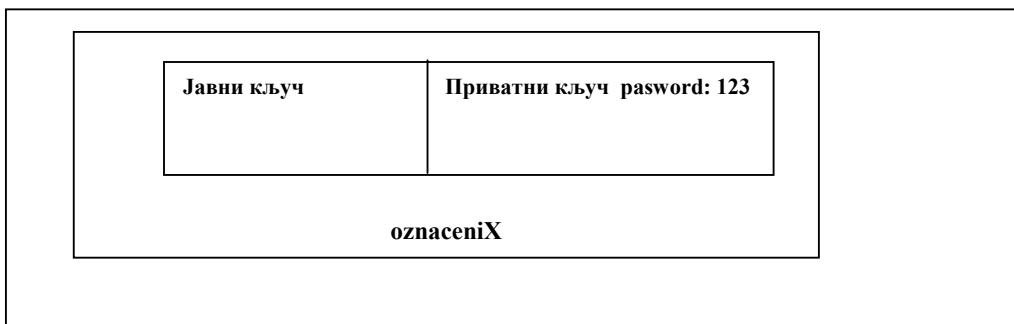
```
jar cvf X.jar X1.class X2.class...,Xn.class
```

Генеришу се кључеви.

```
keytool -genkey -alias oznaceniX --keystore bazaKljuceva -keypass 123 -dname "cn=Vlajic" -storepass 321
```

Наредба *keytool* генерише пар кључева који су идентификовани преко алиаса **OznaceniX**. Кључеви су сачувани у бази (наредба **-keystore**) под називом **bazaKljuceva**. Бази се може приступити преко шифре 321. Бази се одређује шифра преко наредбе **-storepass**.

Приватном кључу може се приступити преко алиса **OznaceniX** и шифре 123. Приватном кључу се одређује шифра преко наредбе **-keypass**.



Означава се *jar* датотека:

```
jarsigner -keystore bazaKljuceva -storepass 321 -keypass 123 -signedjar xx.jar x.jar
oznaceniX
```

Из базе *bazaKljuceva* се узима приватни кључ и уграђује се у *jar* датотеку. На тај начин *jar* датотека бива означена.



Генерирање сертификованог јавног кључа:

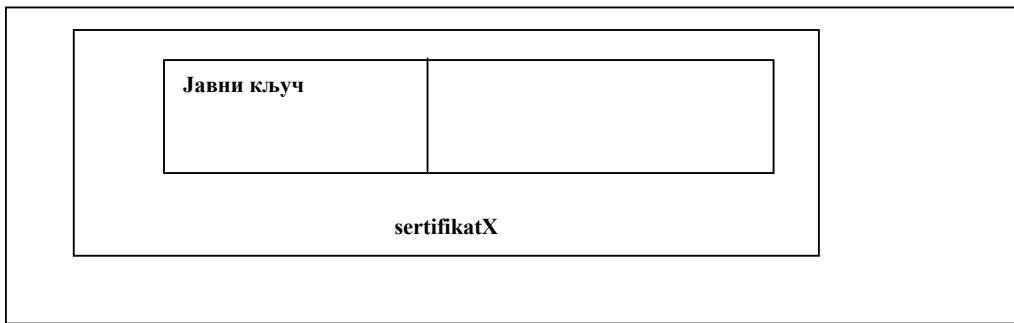
```
keytool -export -keystore bazaKljuceva -storepass 321 -alias oznaceniX -file X.cer
```

Сертификат јавног кључа се чува у датотеци *X.cer* и он се заједно са *xx.jar* датотеком шаље до крајњег корисника.

Крајњи корисник прохвата *X.cer* и *xx.jar* датотеку.

Импортује се сертификат као проверени (*trusted*) сертификат:

```
keytool -import -alias sertifikatX -file X.cer -keystore bazaKljuceva -
storepass 432
```



Креира се *policy* датотека(*X.poli*):

```

keystore "bazaKljuceva";

grant signedBy "sertifikatX" {
    permission java.io.FilePermission "X.ini", "write";
    permission java.io.FilePermission "X.ini", "read";
}

```

Крајњи корисник може да чита и да уписује у датотеку *X.ini* на локалној машини.
Покреће се *appletviewer* ка:

```
appletviewer -J-Djava.security.policy=X.poli X.html
```

Претпоставља се да је креирана датотека *X.html* која позива аплет *X1.class*:

```

<applet code ="X1.class"
        archive = "XX.jar"
        width = 350 height=550
        param name=file value="">
</applet>

```

Аплет ће моћи да изврши операције читања и писања у датотеку уколико сертификованни јавни кључ може у *XX.jar* да верификује приватни кључ.

Пример ОАП1:

1. Датотека OA.java

```

/*
@autor Sinisa Vlajic
* SILAB - Laboratorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/

// Programski zahtev: Napisati program koji ce preko apleta da prihvata serijalizovane
objekte.
// Ogranicenje u programu je sledeće:
// 1. Može se uneti najviše 2 org. jed
// 2. Može se uneti najviše 3 radnika
// 3. Kada se unosi radnik on mora da se poveze sa org. jedinicom koja postoji.
// Na pocetku izvršenja apleta napuniti objekte iz datoteke. Na kraju izvršenja apleta
// zapamtite aplete u datoteci.

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

class Radnik implements Serializable {
    String ImeRadnika; OrgJed oj;
    Radnik(String ImeRadnikal, OrgJed oj1) { ImeRadnika = ImeRadnikal; oj=oj1; }
    Radnik(){}
}

class OrgJed implements Serializable {
    String SifraOrgJed; String Naziv;
    OrgJed(String SifraOrgJed1, String Naziv1) { SifraOrgJed = SifraOrgJed1; Naziv=Naziv1; }
    OrgJed(){}
}

class SlozObjekat implements Serializable {
    Radnik []r=new Radnik[3];
    OrgJed oj []=new OrgJed[2];
    int brojrad=0;
    int brojorg=0;

    SlozObjekat NapuniObjekteIzDatoteke() // {return new SlozObjekat();}
    {
        SlozObjekat so = new SlozObjekat(); // ovde se inicijalizuje.
        try{
            String fileName = "OA12.ini";
            FileInputStream fis = new FileInputStream(fileName);
            ObjectInputStream dos= new ObjectInputStream(fis);
            so=(SlozObjekat) dos.readObject(); // Ako nema datoteke desava se izuzetak
            // tako da se so ne puni ni sa cim i ostaje na pocetnoj inicijalizaciji.
            // Ako postoji datoteka tada se so puni sa zapamcenim vrednostima.
            System.out.println(so.brojrad);
        }
    }
}

```

```

        System.out.println(so.brojorg);
        fis.close();
    } catch(Exception e){System.out.println(e);}
    return so;
}

void ZapamtiSveUDatoteci() {
    try{ String fileName = "OA12.ini";
    FileOutputStream fos = new FileOutputStream(fileName);
    ObjectOutputStream dos= new ObjectOutputStream(fos);
    dos.writeObject(this);
    dos.flush();
    fos.close();
    } catch(Exception e){}
}

public String ZapamtiRad(String imer, String org) {
    if (brojrad < 3) {
        r[brojrad] = new Radnik();
        r[brojrad].ImeRadnika = imer;
        r[brojrad].oj = PronadjiOrgJed(org);
        if (r[brojrad].oj == null)
            return "Ne postoji org.jedinica!!!";
        else {
            brojrad++;
            ZapamtiSveUDatoteci();
            return "Uspesan unos radnika!!!";
        }
    } else
        return "Uneli ste 3 radnika!!!";
}

public String ZapamtiOrg(String sorg, String norg) {
    if (brojorg < 2) {
        oj[brojorg] = new OrgJed();
        oj[brojorg].SifraOrgJed = sorg;
        oj[brojorg].Naziv = norg;
        brojorg++;
        ZapamtiSveUDatoteci();
        return "Uspesan unos org. jedinice!!!";
    } else
        return "Uneli ste 2 org. jedinice!!!";
}

public OrgJed PronadjiOrgJed(String orgjed1) {
    for(int i=0;i<brojorg;i++)
        if (oj[i].SifraOrgJed.equals(orgjed1))
            return oj[i];
    return null;
}

public class OA extends Applet implements ActionListener, ItemListener {
    TextField tsorg,tnorg;
    TextField timer;
    Choice komboorg;
    Button bradnik, borganizacija;
    Label umetak[] = new Label[10];
    int bu =0;
    SlozObjekat so = new SlozObjekat();
    String poruka;

    public void init() {
        String str;
        int duzina;
        so = so.NapuniObjekteIzDatoteke();
        setBackground(Color.yellow);
        setForeground(Color.blue);
        setLayout(new FlowLayout(FlowLayout.LEFT));
        str = "OBRADA ORGANIZACIJE";
        Label lstudent = new Label(str);add(lstudent);Umetak(str.length());
        str = "Sifra organizacije"; duzina =7;
        Label lsorg = new Label(str);add(lsorg);
        tsorg = new TextField(duzina);add(tsorg); Umetak(str.length()+duzina);
        str = "Naziv organizacije";duzina =20;
        Label lnorg = new Label(str);add(lnorg);
        tnorg = new TextField(duzina);add(tnorg);
        Umetak(str.length()+duzina);
    }
}

```

```

borganicacija = new Button("Zapamti organizacija");add(borganizacija);Umetak(0);
str = "OBRADA RADNIKA";
Label lradnik = new Label(str);add(lradnik);Umetak(0);
str = "Ime radnika"; duzina =30;
Label limer = new Label(str); add(limer);
timer = new TextField(duzina);add(timer);Umetak(str.length()+duzina);
str = "Sifra org. jed"; duzina =7;
Label lorg= new Label(str); add(lorg);
komboorg = new Choice();
for(int i=0;i<so.brojorg;i++)
    komboorg.add(so.oj[i].SifraOrgJed);
add(komboorg); Umetak(0);
str= "Zapamti radnik";
bradnik = new Button(str);add(bradnik);
tsorg.addActionListener(this);
tnorg.addActionListener(this);
timer.addActionListener(this);
komboorg.addItemListener(this);
bradnik.addActionListener(this);
borganicacija.addActionListener(this);
}
}

void Umetak(int brojpraznih) {
    int duzinanaj = 42;
    char n[] = new char [duzinanaj];
    for(int i=0;i<duzinanaj-brojpraznih;i++)
        n[i]=' ';
    String s = new String(n,0,duzinanaj-brojpraznih);
    umetak[bu] = new Label(s);add(umetak[bu]);
    bu++;
}
public void paint(Graphics g) {
    g.drawString("Poruka o uspesnom unosu:", 6,300);
    g.drawString("Poruka o uspesnom unosu:" + poruka, 6,300);
    int j=0;
    for(int i=0;i<so.brojrad; i++) {
        g.drawString("Ime radnik:" + so.r[i].ImeRadnika, 6, 300+ (j+=20));
        g.drawString("Org jed:" + so.r[i].oj.Naziv, 6, 300+ (j+=20));
    }
    for(int i=0;i<so.brojorg;i++) {
        g.drawString("Sifra org:" + so.oj[i].SifraOrgJed, 6, 300+ (j+=20));
        g.drawString("Naziv org:" + so.oj[i].Naziv, 6, 300+ (j+=20));
    }
    g.drawString("Poruka o uspesnom unosu:" + poruka, 6,300);
}

public void actionPerformed(ActionEvent ae) {
    String strdugme = ae.getActionCommand();
    if (strdugme.equals("Zapamti radnik"))
        poruka = so.ZapamtiRad(timer.getText(),komboorg.getSelectedItem());
    if (strdugme.equals("Zapamti organizacija"))
        poruka = so.ZapamtiOrg(tsorg.getText(),tnorg.getText());
    repaint();
}

public void itemStateChanged(ItemEvent ie)      { repaint();}

public void stop(){
    so.ZapamtiSveUDatoteci();
}
}

```

2. Датотека OA.html

```

<applet code ="OA.class"
archive = "OOA.jar"
width = 350 height=550
<param name=file value="">
</applet>

```

3. Датотека OA12.ini која је празна.

4. Датотека OA.jp

```

keystore "OAKLIENT";
grant signedBy "OA" {

```

```
permission java.io.FilePermission "OA12.ini", "write";
permission java.io.FilePermission "OA12.ini", "read";
};
```

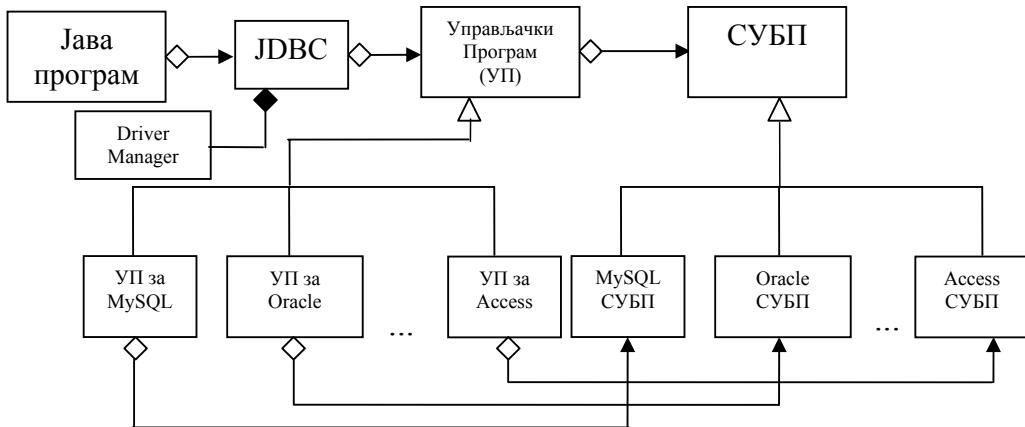
5. Датотека OA.bat која покреће аплет:

```
javac OA.java
jar cvf OA.jar OA.class Radnik.class OrgJed.class SlozObjekat.class
keytool -genkey -alias signOA -keystore OADAT -keypass OAPRIVATE -dname "cn=VLAJIC" -storepass
OAPUBLIC
jarsigner -keystore OADAT -storepass OAPUBLIC -keypass OAPRIVATE -signedjar OOA.jar OA.jar
signOA
keytool -export -keystore OADAT -storepass OAPUBLIC -alias signOA -file OA.cer
keytool -import -alias OA -file OA.cer -keystore OAKLIENT -storepass OAKLPUB
appletviewer -J-Djava.security.policy=OA.jp OA.html
```

Задатак ЗОАП1: Написати сопствени означенчи аплет који ће у датотеци чувати бројеве који се прихватају преко аплета. Омогући да се запамћени бројеви могу прочитати из датотеке и приказати.

6. РАД СА БАЗОМ – JDBC

Повезивање неког програма који је написан Јави и неког од Система за управљање базом података (*MySQL*, *Oracle*, *SQL Server*,..., *MS Access*) ради се преко: а) Јавиног *JDBC* (*Java Database Connectivity*) API-а и б) управљачког програма (драјвера) који се прави посебно за сваки СУБП (Слика СлБП1).



СлБП1: Веза Јава програма са СУБП

Једном написан Јава програм који се извршава над неким Системом за управљање базом података (СУБП) може се извршавати непромењен и над другим СУБП³². Једини предуслов извршења неког Јава програма над неким СУБП јесте постојање управљачког програма за тај СУБП.

6.1 Поступак повезивања Јава програма и СУБП-а

Поступак повезивања Јава програма и базе података изабраног СУБП се изводи у следећим корацима:

- укључивање у Јава програм *JDBC API-а*
- учитавање управљачког програма у Јава програм
- успостављање конекције (везе) између Јава програма и базе података изабраног СУБП

У даљем тексту ће наведени кораци бити детаљно објашњени.

a) укључивање у Јава програм *JDBC API-а*

Укључивање *JDBC API-а* у Јава програм се ради преко следеће наредбе³³:

```
import java.sql.*;
```

б) учитавање управљачког програма у Јава програм

Учитавање управљачког програма у Јава програм се ради преко следеће наредбе:

```
Class.forName("com.mysql.jdbc.Driver");
```

³² У суштини се Јава програм извршава над неком базом података која се налази унутар изабраног СУБП. Нпр. Јава програм се повезује са базом података *Student* која се налази унутар *MySQL* СУБП.

³³ У пакету *java.sql* налазе се класе које се користе у раду са базама података СУБП.

```
/* Primer BP1: Pokazati na jednom primeru kako se vrši ucitavanje drajvera za MySQL SUBP i za
MS Access SUBP */

class BP1 {
    public static void main(String[] args) {
        try {
            // Ucitavanje drajvera za MySQL bazu
            Class.forName("com.mysql.jdbc.Driver");
            // Ucitavanje drajvera za MS Access bazu
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println("Upravljacki programi su ucitani!");
        } catch (ClassNotFoundException cnfe) {
            System.out.println("Nije ucitan upravljacki program: " + cnfe);
        }
    }
}
```

У даљем тексту покушаћемо да детаљније објаснимо шта је све потребно урадити у окружењу Јава програма како би се успешно извршила наредба:

```
Class.forName("com.mysql.jdbc.Driver");
```

Управљачки програм је у суштини Јавина *class* датотека која се обично налази у некој *jar* датотеци. Тако се на пример један од *MySQL* управљачких програма (*Driver.class*) налази у датотеци:

mysql-connector-java-3.1.12-bin.jar

Датотека *Driver.class* налази се унутар *mysql-connector-java-3.1.12-bin.jar*³⁴ датотеке у пакету *com.mysql.jdbc*³⁵

На основу наведеног може се закључити да:

1. параметар *forName* методе класе *Class*:

```
"com.mysql.jdbc.Driver"
```

у суштини представља путању (*com.mysql.jdbc*) унутар *mysql-connector-java-3.1.12-bin.jar* датотеке до *Driver.class* датотеке, која је, наглашавамо, **управљачки програм**.

2. уколико желимо да учитамо управљачки програм у Јава програм потребно је повезати Јава програм са *mysql-connector-java-3.1.12-bin.jar* датотеком.

Повезивање Јава програма са *mysql-connector-java-3.1.12-bin.jar* датотеком се ради на стандардан начин³⁶, као и са било којом *jar* датотеком.

ц) успостављање конекције (везе) између Јава програма и базе података изабраног СУБП. Успостављање конекције се ради помоћу *JDBC DriverManager* класе. Као резултат успостављања конекције са базом података преко *JDBC DriverManager* класе добија се објекат класе *Connection* који чува конекцију ка бази података.

³⁴ Драјвер за *MS Access* се налази у *rt.jar* датотеци.

³⁵ Када би се датотека *mysql-connector-java-3.1.12-bin.jar* распаковала она би креирала фолдер *jdbc* који се налази испод фолдера *com/mysql*. Прецизније речено добија се следећа хијерархија фолдера: *com/mysql/jdbc*. У фолдеру *jdbc* се налази управљачки програм (*Driver.class*) за *MySQL* СУБП.

³⁶ Уколико се ради са неким од једноставнијих Јавиних развојних окружења (нпр. *TextPad*) тада је потребно у системској променљивој *CLASSPATH* да се се наведе пут до наведене *jar* датотеке: *C:\Install\MySQL5.0\mysql-connector-java-3.1.12\mysql-connector-java-3.1.12\mysql-connector-java-3.1.12-bin.jar*;

Уколико се ради у неком од сложенијих Јавиних развојних окружења (нпр. *NetBeans*) могуће је у Јава програм на једноставан начин (*properties/AddJar*) укључити наведену *jar* датотеку .

```
/*Primer BP21: Pokazati kako se uspostavlja veza (konekcija) sa MySQL bazom podataka.
*/
import java.sql.*;

class BP21 {
    public static void main(String[] args) {
        try {
            String dbUrl = "jdbc:mysql://127.0.0.1:3306/student";
            String user = "root";
            String pass = "root";
            Class.forName("com.mysql.jdbc.Driver");
            Connection naredba = DriverManager.getConnection(dbUrl, user, pass);
            System.out.println("Uspostavljena je konekcija izmedju driver manager-a i baze");
        } catch (ClassNotFoundException cnfe) {
            System.out.println("Nije ucitan upravljacki program: " + cnfe);
        } catch (SQLException sqle) {
            System.out.println("Greska: " + sqle);
        }
    }
}
```

Објашњење најважнијих делова програма BP21:

Када се учита *MySQL* управљачки програм у Јава програм преко наредбе:

```
Class.forName("com.mysql.jdbc.Driver");
```

тада управљачки програм добија симболички назив преко кога се он касније позива³⁷. У случају наведене наредбе управљачки програм добија симболички назив³⁸: `jdbc:mysql`

У примеру BP21 се налази наредба:

```
String dbUrl="jdbc:mysql://127.0.0.1:3306/student";
```

у којој променљива `dbUrl` добија следећу *url* адресу:

```
"jdbc:mysql://127.0.0.1:3306/student"
```

Наведена адреса се може декомпоновати на следеће делове:

Назив управљачког програма (`jdbc:mysql`) који је учитан преко наредбе:

```
Class.forName("com.mysql.jdbc.Driver");
```

у Јава програм.

IP адреса (`127.0.0.1`) машине на којој се налази *MySQL* СУБП.

Порт (`3306`) на коме је подигнут *MySQL* СУБП.

Име базе података (`student`) са којом Јава програм успоставља конекцију преко наредбе:

```
Connection CONECTION=DriverManager.getConnection(dbUrl,user,pass);
```

На основу наведеног се може закључити да наредба:

```
String dbUrl="jdbc:mysql://127.0.0.1:3306/student";
```

даје адресу до базе података **student** која се налази на *MySQL* серверу који се налази на адреси: **127.0.0.1:3306** преко управљачког програма чији је симболички назив: `jdbc:mysql`. Наведени однос између Јава програма B21, управљачког програма се види на слици СлБП2.

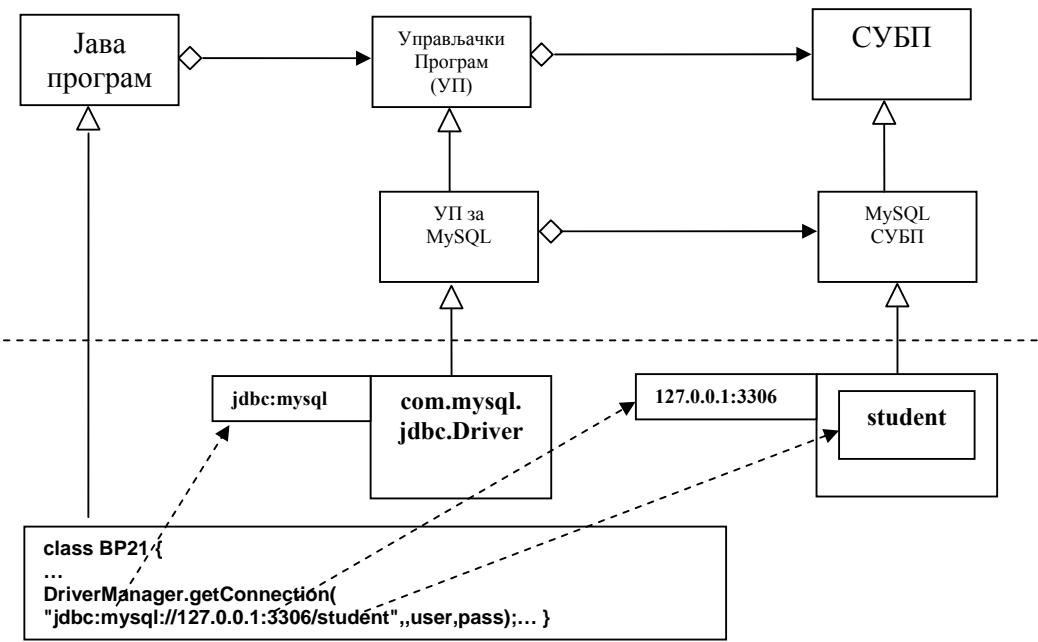
³⁷ Тада симболички назив се чува у *Driver Manager*-у. Када се *Driver Manager* напуни са симболичким називом неког драјвера, за такав драјвер кажемо да је регистрован.

³⁸ Када се учита *MS Access* управљачки програм у Јава програм преко наредбе:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

тада управљачки програм добија симболички назив:

```
jdbc:odbc
```



СЛБП2: Однос између Јава програма, управљачког програма и базе података

На крају се помоћу наредбе:

```
Connection CONECTION=DriverManager.getConnection(dbUrl,user,pass);
```

успоставља конекција са базом података између Јава програма и изабране базе података (*student*). Када се позове метода `getConneќtion()`, она итеративно пролази кроз драјвере регистроване у *DriverManager*-у и испитује да ли постоји драјвер са задатим називом. Уколико се пронађе драјвер, класа *DriverManager* прави објекат *Connection*, који успоставља везу са базом података, помоћу пронађеног драјвера.

Задатак ВРZ1: Креирати базу података *Prodaja* у MySQL СУБП и остварити конекцију са њом. База података *Prodaja* има табелу *Racun* која има следеће атрибути: *BrojRacuna* типа *String*, *NazivPartnera* типа *String*, *UkupnaVrednost* типа *double*, *Obradjen* типа *boolean*, *Storaniran* типа *boolean*.

6.2 Поступак извршења операција над базом података СУБП

Поступак извршења операција над СУБП се изводи у следећим корацима:

a) прављење објекта класе statement

Помоћу конекције која је успостављена преко наредбе:

```
Connection konekcija =DriverManager.getConnection(dbUrl,user,pass);
```

се прави објекат наредба класе Statement преко:

```
Statement naredba=konekcija.createStatement();
```

Помоћу објекта наредба се изводе операције над базом података преко:

```
naredba.executeQuery(upit);
```

У примеру BP31 ће се видети како се изводи операција *SELECT* над базом података.

```
/* Primer BP31: Napisati program kojim se prikazuje trenutni sadrzaj tabele Student.
Baza podataka, u kojoj se nalazi tabela Student, je realizovana preko MySQL i MS Access SUBP39*/
import java.sql.*;

class BP31 {
    public static void main(String[] args) {
        try {
            String dbUrl = new String();
            String user = "root";
            String pass = "root";

            if (args[0].equals("1")) {
                dbUrl = "jdbc:odbc:student";
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            }

            if (args[0].equals("2")) {
                dbUrl = "jdbc:mysql://127.0.0.1:3306/student";
                Class.forName("com.mysql.jdbc.Driver");
            }

            Connection konekcija = DriverManager.getConnection(dbUrl, user, pass);
            Statement naredba = konekcija.createStatement();
            String upit = "SELECT brind,ime,prezime FROM Student";
            ResultSet rs = null;
            try {
                rs = naredba.executeQuery(upit);
            } catch (SQLException sqle) {
                System.out.println("Greska u izvr. upita: " + sqle);
            }

            System.out.println("Trenutan izgled tabele studenata!");
            while (rs.next()) {
                System.out.println(rs.getString("brind") + " " + rs.getString("ime")
                    + " " + rs.getString("prezime"));
            }

            naredba.close();
            konekcija.close();
        } catch (ClassNotFoundException cnfe) {
            System.out.println("Nije ucitan upravljacki program: " + cnfe);
        } catch (SecurityException se) {
            System.out.println("Nedozvoljena operacija: " + se);
        } catch (SQLException sqle) {
            System.out.println("Greska konekcije: " + sqle);
        }
    }
}
```

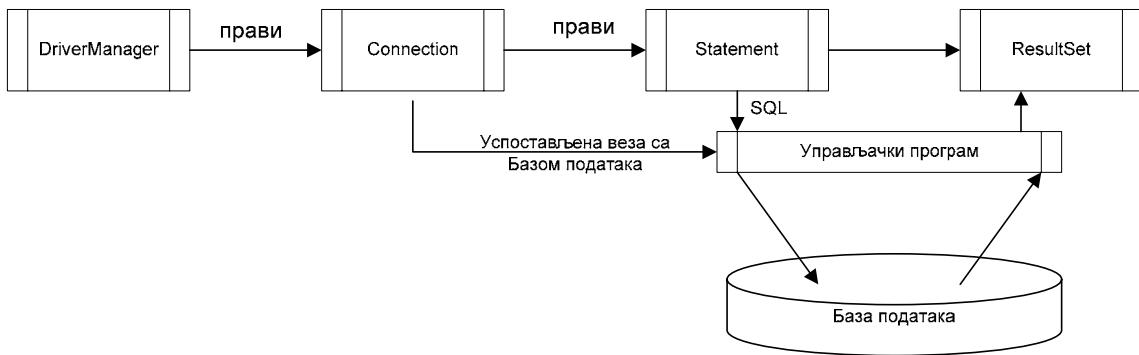
У наведеном примеру се упит:

"SELECT brind,ime,prezime FROM Student"

шаље као параметар методе `executeQuery` која се извршава над објектом наредба. Резултат те наредбе се чува у `ResultSet`-у `rs`. У `ResultSet`-у се чувају сви слогови табеле *Student*. Кроз наведени `ResultSet` се пролази и приказује се његов садржај.

На слици СЛБПЗ се виде одговорности класа које учествују у поступку извршења операције над базом података..

³⁹ Уколико се жели из Јава програма приступити некој *MS Access* бази података која се налази на локалном рачунару, мора се приступити регистрацији базе података. Регистрација базе се на *Windows* оперативним системима обавља у *ODBC Data Source Administrator*-у који се налази на путањи: **Start -> Control Panel -> Administrative Tools -> Data Sources**.



СлБП3: Поступак извршења операције над базом података

Уколико желимо да покренемо програм BP31 преко bat датотеке њен садржај је:

```

set classpath=C:\Install\MySQL5.0\mysql-connector-java-3.1.12\mysql-connector-java-3.1.12\mysql-connector-java-3.1.12-bin.jar;
C:
CD \Predavanja\JDBCPetiCas\Zadaci
javac BP31.java
java BP31 2
PAUSE

```

```

/* Primer BP4: Napisati program koji u tabelu Student sa atributima broj indeksa, ime i prezime unosi novog studenta sa brojem indeksa 01/06 ,cije je ime Pera , a prezime Peric.*/
import java.sql.*;

class BP4 {
    public static void main(String[] args) {
        try {
            String dbUrl = "jdbc:odbc:student";
            String user = "";
            String pass = "";
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection CONECTION = DriverManager.getConnection(dbUrl, user, pass);
            String upit = "INSERT INTO Student(brind,ime,prezime) VALUES (?,?,?)";
            PreparedStatement PSTATEMENT = CONECTION.prepareStatement(upit);
            PSTATEMENT.setString(1, new String("01/06"));
            PSTATEMENT.setString(2, new String("Pera"));
            PSTATEMENT.setString(3, new String("Peric"));
            try {
                PSTATEMENT.executeUpdate();
                System.out.println("Novi student je zapamcen u bazi");
            } catch (SQLException e) {
                System.out.println("Izuzetak: " + e);
            }
            PSTATEMENT.close();
            CONECTION.close();
        } catch (ClassNotFoundException cnfe) {
            System.out.println("Nije ucitan upravljacki program: " + cnfe);
        } catch (SQLException sqle) {
            System.out.println("Greska kod konekcije: " + sqle);
        }
    }
}

```

```

/* Primer BP5: Napisati program koji ce da prihvata i cuva podatke u tabelu Student. Tabela se cuva u bazi koja je realizovana u okviru MySql SUBP. Naziv baze je Student. Program treba da bude tronivojski, sto znaci da treba razdvojiti nivoe korisnickog interfejsa, poslovne logike i baze podataka.
*/
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.*;

import javax.swing.*;
import org.netbeans.lib.awtextra.AbsoluteConstraints;

```

```
import org.netbeans.lib.awtextra.AbsoluteLayout;

/** *****PREZENTACIONI NIVO PROGRAMA***** */
public class BP5GUI extends JFrame {
    // Labela koja sadrzi naziv ekranske forme koja se otvara.
    private JLabel LNazivForme;

    // Polja preko kojih se obradjuju podaci.
    private JFormattedTextField PprojInd;
    private JFormattedTextField PIime;
    private JFormattedTextField PPrezime;

    // Labele koje opisuju polja za obradu podataka.
    private JLabel LbrojInd;
    private JLabel LIime;
    private JLabel LPrezime;

    // Dugme preko koga se poziva sistemska operacija.
    private JButton BZapamti;

    // Glavni program
    public static void main(String args[]) {
        BP5GUI gui = new BP5GUI();
        gui.show();
    }

    // 1. Konstruktor ekranske forme
    public BP5GUI() {
        KreirajKomponenteEkranskeForme(); // 1.1
        PokreniMenadzeraRasporedaKomponeti(); // 1.2
        PostaviImeForme(); // 1.3
        PostaviPoljaZaPrihvataPodataka(); // 1.4
        PostaviLabeleZaPrihvataPodataka(); // 1.5
        PostaviDugmeZapamti(); // 1.6
        pack();
    }

    // 1.1 Kreiranje i inicializacija komponenti ekranske forme
    void KreirajKomponenteEkranskeForme() {
        LNazivForme = new JLabel();
        PprojInd = new JFormattedTextField();
        PIime = new JFormattedTextField();
        PPrezime = new JFormattedTextField();

        LbrojInd = new JLabel();
        LIime = new JLabel();
        LPrezime = new JLabel();
        BZapamti = new JButton();
    }

    // 1.2 Kreiranje menadzera rasporeda komponenti i njegovo dodeljivanje do
    // kontejnera okvira(JFrame komponente).
    void PokreniMenadzeraRasporedaKomponeti() {
        getContentPane().setLayout(new AbsoluteLayout());
    }

    // 1.3 Odredivanje naslovnog teksta i njegovo dodeljivanje do kontejnera
    // okvira.
    void PostaviImeForme() {
        LNazivForme.setFont(new Font("Times New Roman", 1, 12));
        LNazivForme.setText("UNOS STUDENATA");
        getContentPane().add(LNazivForme, new AbsoluteConstraints(20, 10, -1, -1));
    }

    // 1.4
    void PostaviPoljaZaPrihvataPodataka() { // Dodeljivanje pocetne vrednosti i
        // formata polja.
        PprojInd.setValue(new String(""));
        PIime.setValue(new String(""));
        PPrezime.setValue(new String(""));
        // Polja se dodaju kontejneru okvira (JFrame).
        getContentPane().add(PprojInd, new AbsoluteConstraints(120, 70, 100, -1));
        getContentPane().add(PIime, new AbsoluteConstraints(120, 100, 100, -1));
        getContentPane().add(PPrezime, new AbsoluteConstraints(120, 130, 100, -1));
    }

    // 1.5
    void PostaviLabeleZaPrihvataPodataka() {
        LbrojInd.setText("Broj indeksa:");
        getContentPane().add(LbrojInd, new AbsoluteConstraints(20, 70, 100, -1));
    }
}
```

```

LIme.setText("Ime:");
getContentPane().add(LIme, new AbsoluteConstraints(20, 100, 100, -1));
LPrezime.setText("Prezime:");
getContentPane().add(LPrezime, new AbsoluteConstraints(20, 130, 100, -1));
}

// *****
// 1.6
// Deklarisanje i kreiranje objekta koji je odgovoran za logiku programa.
Logika l = new Logika();

void PostaviDugmeZapamti() {
    BZapamti.setText("Zapamti");
    BZapamti.addActionListener(new ActionListener() {
        /**
         * DOGADJAJ KOJI INICIRA POZIV SISTEMSKE OPERACIJE
         */
        public void actionPerformed(ActionEvent evt) {
            try {
                /**
                 * POZIV SISTEMSKE OPERACIJE
                 */
                String slog = "" + PbrojInd.getValue() + ", "
                    + (String) PIIme.getValue() + ',' + (String) PPrezime.getValue()
                    + "!";
                l.pamtiSlog(slog);
            } catch (Exception e) {
                l.postaviPoruku("Nije zapamcen slog!");
            }
            PrikaziPoruku();
            show();
        }
    });
    // Kraj addActionListener metode
    /**
     */
}

getContentPane().add(BZapamti, new AbsoluteConstraints(260, 60, -1, -1));
}

void PrikaziPoruku() {
    if (!l.signal) {
        Poruka p = new Poruka(this);
        p.prikazi(l.poruka);
        p.show();
    }
}
}

class Poruka extends JDialog {
    public Poruka(JFrame roditelj) {
        super(roditelj, "UPOZORENJE", true);
    }

    void prikazi(String poruka) {
        Box b = Box.createVerticalBox();
        b.add(new JLabel(poruka));
        getContentPane().add(b);
        setSize(350, 70);
    }
}

// Logicki nivo programa i nivo baze podataka, odnosno klase Logika i
// BazaPodataka
// nalaze se u datoteci BP5LogikaiBaza.java

/**
 * *****LOGICKI NIVO
 * PROGRAMA*****
 */
class Logika {
    String poruka = "";
    boolean signal;
    BazaPodataka bp;

    public boolean pamtiSlog(String slog) {
        bp = new BazaPodataka();
        if (!bp.otvoriBazu()) {
            poruka = bp.poruka;
            return false;
        }
        if (!bp.pamtiSlog("Student", slog)) {
            poruka = bp.poruka;
            return false;
        }
    }
}

```

```

        if (!bp.zatvoriBazu()) {
            poruka = bp.poruka;
            return false;
        }
        poruka = bp.poruka;
        return true;
    }

    void postaviPoruku(String pom) {
        poruka = new String(pom);
        signal = false;
    }

}

/***
 * *****NIVO BAZE
 * PODATAKA*****
 */
class BazaPodataka {
    Connection con;
    String poruka;

    boolean otvoriBazu() {
        String dbUrl = new String();
        String user = "root";
        String pass = "root";
        try {
            dbUrl = "jdbc:mysql://127.0.0.1:3306/student?useUnicode=true&characterEncoding=UTF-8";
            // VAŽNA NAPOMENA: Na ovaj način je omogućeno da se pamte cirilična i
            // latinična
            // slova u MySQL bazi. Na strani baze takodje je potrebno da atributi
            // tabele budu
            // vezani za charset UTF-8.
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(dbUrl, user, pass);
        } catch (ClassNotFoundException cnfe) {
            poruka = "Nije ucitan upravljacki program: " + cnfe;
            System.out.println(poruka);
            return false;
        } catch (SQLException sqle) {
            poruka = "Greska kod konekcije: " + sqle;
            System.out.println(poruka);
            return false;
        }
        return true;
    }

    boolean pamtiSlog(String imeTabele, String vrednostiAtributa) {
        String upit;
        try {
            Statement st = con.createStatement();
            upit = "INSERT INTO " + imeTabele + " VALUES (" + vrednostiAtributa + ")";
            st.executeUpdate(upit);
            poruka = "Slog je uspesno zapamcen u bazi";
            System.out.println(poruka);
            st.close();
        } catch (SQLException esql) {
            poruka = "Nije uspesno zapamcen slog u bazi: " + esql;
            System.out.println(poruka);
            return false;
        }
        return true;
    }

    boolean zatvoriBazu() {
        try {
            con.close();
        } catch (Exception e) {
            poruka = "Nije uspesno zatvorena baza:" + e;
            System.out.println(poruka);
            return false;
        }
        return true;
    }
}

```

```
/* Primer BP6: Napisati program koji će da brise slogove iz
tabela Student. Tabela se cuva u bazi koja je realizovana
u okviru MySQL SUBP. Naziv baze je Student. Program treba da bude
tronivojski, što znači da treba razdvojiti nivoe korisnickog interfejsa,
poslovne logike i baze podataka.
*/
```

Наведени задатак је сличан предходном задатку. У наставку ће бити дате само разлике везане за овај програм.

```
// Prezentacioni nivo programa, odnosno klase BP6GUI i Poruka nalaze se u
// datoteci BP6GUI.java

/*********************PREZENTACIONI NIVO PROGRAMA*****/
public class BP6GUI extends JFrame
{
...
    // Dugme preko koga se poziva sistemska operacija.
    private JButton BObrixi;

    // Glavni program
    public static void main(String args[])
    { BP6GUI   gui = new BP6GUI ();
        gui.show();
    }

    // 1. Konstruktor ekranske forme
    public BP6GUI   ()
    {
...
        PostaviDugmeObrisi();
        pack();
    }

    // 1.6
    // Deklarisanje i kreiranje objekta koji je odgovoran za logiku programa.
    Logika l = new Logika();
    void PostaviDugmeObrisi()
    { BObrixi.setText("Obrisi");
        BObrixi.addActionListener(new ActionListener() {
    /***** DOGADJAJ KOJI INICIRA POZIV SISTEMSKE OPERACIJE *****
        public void actionPerformed(ActionEvent evt)
        {
            try { /***POZIV SISTEMSKE OPERACIJE***/
                String uslov = "BrInd = '" + PprojInd.getValue() + "'";
                l.obrisiSlog(uslov);
            } catch(Exception e) {l.postaviPoruku("Nije obrisan slog!");}
            PrikaziPoruku();
            show();
        }
    });
    /*****getContentPane().add(BObrixi, new AbsoluteConstraints(260, 60, -1, -1));
}

class Poruka extends JDialog
{...

// Logicki nivo programa i nivo baze podataka, odnosno klase Logika i BazaPodataka
//nalaze se u datoteci BP6LogikaiBaza.java

...
/*********************LOGICKI NIVO PROGRAMA*****/

class Logika
{
    String poruka="";
    boolean signal;
    BazaPodataka bp;
    public boolean obrisiSlog(String uslov)
    { bp = new BazaPodataka();

        if (!bp.otvoriBazu()) { poruka = bp.poruka;return false;}
        if (!bp.brisiSlog("Student",uslov)) { poruka = bp.poruka;return false; }
        if(!bp.zatvoriBazu())      {poruka = bp.poruka;return false;}
        poruka = bp.poruka;
```

```

        return true;
    }

void postaviPoruku(String pom)
{ poruka = new String(pom);
  signal = false;
}
}

/*****NIVO BAZE PODATAKA*****/


class BazaPodataka
{ Connection con;
  String poruka;

  boolean otvoriBazu()
  {...}

  public boolean brišiSlog(String imeTabele, String uslovZaNadjislog)
  { String upit;
    try { Statement st;
      st = con.createStatement();
      upit ="DELETE FROM " + imeTabele + " WHERE " + uslovZaNadjislog;
      st.executeUpdate(upit);
      poruka = "Slog je uspesno obrisan";
      System.out.println(poruka);
      st.close();
    } catch(SQLException esql) { poruka = "Nije uspesno obrisan slog u bazi: " +
esql;
                                System.out.println(poruka); return
false; }
    return true;
  }

  boolean zatvoritiBazu() { ... }
}

```

Задатак BPZ2: Омогућити памћење, промену и брисање слогова табеле *Račun* базе *Prodaja* преко стандардног улаза.

Задатак BPZ3: Приказати све слогове табеле *Račun* базе *Prodaja*.

Задатак BPZ4: Омогућити памћење, промену и брисање слогова табеле *Račun* базе *Prodaja* преко *Swing GUI-a*. Програм треба да буде тронивојски.

6.3 Примери SQL упита над MySQL СУБП

6.3.1 Дефиниција података (data definition)

Пример CRDB1: Направити базу података по имениу *racun*. Подразумевани формат података треба да буде **UTF8**.

```
CREATE DATABASE racun
CHARACTER SET UTF8
```

Пример AIDB1: Код базе података *racun* променити формат податка на *Latin1*.

```
ALTER DATABASE racun
CHARACTER SET Latin1
```

Пример DRDB1: Обрисати базу података *racun1* ако постоји.

```
DROP DATABASE IF EXISTS racun1
```

Пример CRTAB1: Направити табеле:

racun са атрибутима *BrojRacuna*, *NazivPartnera*, *UkupnaVrednost*, *Obradjeni* и *Storniran*. Примарни кључ је атрибут *BrojRačuna*.

stavkaracuna са атрибутима *BrojRacuna*, *RB*, *SifraProizvoda*, *Kolicina*, *ProdajnaCena* и *ProdajnaVrednost*. Примарни кључ је *BrojRačuna* и *RB*.

```

CREATE TABLE `racun`.`racun` (
  `BrojRacuna` VARCHAR(10) NOT NULL,
  `NazivPartnera` VARCHAR(50) NOT NULL,
  `UkupnaVrednost` DOUBLE NOT NULL,
  `Obradjeno` ENUM('N','Y') NOT NULL DEFAULT 'N',
  `Storaniran` ENUM('N','Y') NOT NULL DEFAULT 'N',
  PRIMARY KEY(`BrojRacuna`)
)
CREATE TABLE `racun`.`stavkaracuna` (
  `BrojRacuna` VARCHAR(10) NOT NULL,
  `RB` INTEGER NOT NULL,
  `SifraProizvoda` VARCHAR(20) NOT NULL,
  `Kolicina` INTEGER NOT NULL DEFAULT 0,
  `ProdajnaCena` DOUBLE NOT NULL DEFAULT 0,
  `ProdajnaVrednost` DOUBLE NOT NULL DEFAULT 0,
  PRIMARY KEY(`BrojRacuna`, `RB`)
)

```

Пример RNTAB1: Променити назив табеле stavkaracuna у stavkaracunal.

```
RENAME TABLE stavkaracuna TO stavkaracunal
```

Напомена: После промене назива табеле треба вратити стари назив како би могли остални примери да се тестирају:

```
RENAME TABLE stavkaracunal TO stavkaracuna
```

Пример ALTAB1: Додајте у табелу **stavkaracuna** атрибут JedinicaMere који је знаковног типа дужине 4, који ће бити постављен после атрибута SifraProizvoda.

```
ALTER TABLE stavkaracuna
ADD COLUMN JedinicaMere VARCHAR(4) AFTER SifraProizvoda
```

Пример ALDB2: Обрисати у табели **stavkaracuna** атрибут SifraProizvoda.

```
ALTER TABLE stavkaracuna
DROP COLUMN JedinicaMere
```

Пример DRTAB1: Креирати табелу **test** са атрибутом **testatribut** и након тога је обрисати.

```
CREATE TABLE `racun`.`test` (
  `testatribut` VARCHAR(10) NOT NULL,
)
DROP TABLE `racun`.`test`
```

Пример CRIND1: Поставити индекс над атрибутом **NazivPartnera** табеле **racun**. Прогласити наведени атрибут да буде примарни кључ.

```
CREATE UNIQUE INDEX IndNazPart ON racun(NazivPartnera)
```

Напомена: Индексирањем се повећава брзина приступа до слогова преко атрибута који је индексиран.

Пример DRIND1: Уклонити индекс над атрибутом **NazivPartnera** табеле **racun**.

```
DROP INDEX IndNazPart ON racun
```

6.3.2 Манипулација подацима (data manipulation)

Пример Insert1: Унос три слога рачуна

```
INSERT INTO Racun
VALUES ('1', 'Meridian invest D.O.O', 0, 'N', 'N');
```

```
-----
INSERT INTO Racun
VALUES ('2', 'Perihard inženjering', 0, 'N', 'N');
-----
INSERT INTO Racun
VALUES ('3', 'Sajam', 0, 'N', 'N');
-----
INSERT INTO StavkaRacuna
VALUES ('1', '1','pr1',5,45.00,0);
-----
INSERT INTO StavkaRacuna
VALUES ('1', '2','pr2',1,45.00,0);
-----
INSERT INTO StavkaRacuna
VALUES ('1', '3','pr3',2,56.34,0);
INSERT INTO StavkaRacuna
VALUES ('2', '1','pr1',7,12.56,0);
-----
INSERT INTO StavkaRacuna
VALUES ('2', '2','pr2',7,12.00,0);
-----
INSERT INTO StavkaRacuna
VALUES ('3', '1','pr1',3,45.00,0);
-----
```

Пример Update1⁴⁰: Промени рачуне код којих је назив партнера “Sajam”. Уместо тога ставити “Beogradski sajam”

```
UPDATE Racun SET NazivPartnera = "Beogradski sajam"
WHERE NazivPartnera="Sajam";
```

Пример Update2: Израчунати и запамтити продајне вредности ставки свих рачуна по формулама: ProdajnaVrednost = ProdajnaCena * Kolicina
 UPDATE StavkaRacuna SET ProdajnaVrednost = ProdajnaCena*Kolicina

Пример Del1: Обрисати рачун број 3.

```
DELETE
FROM Racun
WHERE BrojRacuna= "3";
```

Пример Sel1: Приказивање свих слогова за све атрибути навођењем имена атрибута -
 Приказати све рачуне

```
SELECT * FROM racun;
```

BrojRacuna	NazivPartnera	UkupnaVrednost	Obradjen	Storniran
1	Meridian invest D.O.O	0	N	N
2	Perihard inzenjering	0	N	N

Пример Sel2: Приказивање свих слогова за све атрибути преко * - Приказати све рачуне

```
SELECT * FROM racun
```

BrojRacuna	NazivPartnera	UkupnaVrednost	Obradjen	Storniran
1	Meridian invest D.O.O	0	N	N

⁴⁰ Може и овако:

```
UPDATE Racun SET Racun.NazivPartnera = "Beogradski sajam"
WHERE Racun.NazivPartnera="Sajam";
```

2	Perihard inzenjering	0	N	N
---	-------------------------	---	---	---

Пример Sel3: Приказивање свих слогова за један или неколико атрибута (не свих) навођењем имена атрибута - Приказати бројеве рачуна и називе партнера за све рачуне

SELECT BrojRacuna, NazivPartnера FROM racun;

BrojRacuna	NazivPartnера
1	Meridian invest D.O.O
2	Perihard inzenjering

Пример Dist1: Приказивање различитих слогова табеле - Приказати различите бројеве рачуна са ставке рачуна.

SELECT DISTINCT BrojRacuna FROM StavkaRacuna;

BrojRacuna
1

Пример Dist2: Приказивање различитих слогова табеле - Приказати различите редне бројеве рачуна.

SELECT DISTINCT rb FROM StavkaRacuna;

RB
1
2
3
4

Пример Sel4: Приказивање слогова табеле под неким условом - Приказати податке о рачуну чији је број 2.

SELECT * FROM racun WHERE BrojRacuna = "2"

BrojRacuna	NazivPartnера	UkupnaVrednost	Obradjen	Storniran
2	Perihard inzenjering	0	N	N

Пример OrdBy1: Приказ слогова табеле у сортираном редоследу - Приказати редне бројеве ставки рачуна, у опадајућем редоследу, за рачун чији је број 1.

SELECT * FROM StavkaRacuna WHERE BrojRacuna="1" ORDER BY RB DESC;

BrojRacuna	RB	SifraProizvoda	Kolicina	ProdajnaCena	ProdajnaVrednost
1	3	pr3	2	56.34	112.68
1	2	pr2	1	45	45
1	1	pr1	5	45	225

Пример Max1: Функције за израчунавање сумарних информација - Наћи највећу количину код ставки рачуна чији је број 1.

SELECT Max(Kolicina) AS MaxKolicina
FROM StavkaRacuna
WHERE StavkaRacuna.BrojRacuna="1";

MaxKolicina
5

Пример Sum1: Функције за израчунавање сумарних информација - Приказати укупну вредност рачуна чији је број 1.

SELECT Sum(ProdajnaVrednost) AS UkupnaVrednost
FROM StavkaRacuna
WHERE StavkaRacuna.BrojRacuna="1";

UkupnaVrednost
382.68

Пример Count1: Функција Count за рачунање броја слогова у табели - Нади број ставки рачуна чији је број 1.

```
SELECT Count(*) AS BrojStavki
FROM StavkaRacuna
WHERE BrojRacuna="1";
```

BrojStavki
3

Пример Count2: Функција Count за рачунање броја слогова у табели - Нади број ставки рачуна чији је број 1 по атрибуту ProdajnaCena.

```
SELECT Count(ProdajnaCena) AS BrojStavki
FROM StavkaRacuna
WHERE BrojRacuna="1";
```

BrojStavki
3

Пример Count3: Функција Count за рачунање броја слогова у табели - Нади број различитих ставки рачуна чији је број 1 по атрибуту ProdajnaCena.

```
SELECT Count(Distinct ProdajnaCena) AS BrojStavki
FROM StavkaRacuna
WHERE BrojRacuna="1";
```

BrojStavki
2

Рачуна се број слогова који имају разлиčиту цену (56.34,45.00).

Пример GroupBy1: Груписање слогова табеле по једном или више атрибута - Приказати колико је продато комада за сваку групу (врсту) производа.

Да би се схватио наведени пример потребно је анализирати табелу *StavkaRacuna*.

BrojRacuna	RB	SifraProizvoda	Kolicina	ProdajnaCena	ProdajnaVrednost
1	1	pr1	5	45	225
1	2	pr2	1	45	45
1	3	pr3	2	56.34	112.68
2	1	pr1	7	12.56	87.92
2	2	pr2	7	12	84
3	1	pr1	3	45	135

```
SELECT SifraProizvoda,Sum(Kolicina) as UkupnaKolicina FROM stavkaracuna
Group By SifraProizvoda
```

SifraProizvoda	UkupnaKolicina
pr1	15
pr2	8
pr3	2

Пример GroupBy2: Груписање слогова табеле по једном или више атрибута - За сваки рачун показати колико је продато производа (сумарно).

```
SELECT BrojRacuna,Sum(Kolicina) as UkupnaKolicina FROM stavkaracuna Group
By BrojRacuna
```

BrojRacuna	UkupnaKolicina
1	8

2	14
3	3

Пример GroupBy3: Груписање слогова табеле по једном или више атрибута - За сваки рачун одредити број ставки.

```
SELECT BrojRacuna, Count(*) as BrojStavki FROM stavkaracuna Group By
BrojRacuna
```

BrojRacuna	BrojStavki
1	3
2	2
3	1

Пример GroupBy4: Груписање слогова табеле по једном или више атрибута - За сваки производ одредити рачуне. Рачуне сортирати по шифри производа у растућем редоследу и количини у опадајућем редоследу.

```
SELECT SifraProizvoda, BrojRacuna, Kolicina
FROM stavkaracuna
GROUP BY SifraProizvoda, BrojRacuna
ORDER BY SifraProizvoda, Kolicina Desc
```

SifraProizvoda	BrojRacuna	Kolicina
pr1	2	7
pr1	1	5
pr1	3	3
pr2	2	7
pr2	1	1
pr3	1	2

Пример Having1: Селекција слогова преко Having наредбе - Приказати рачуне који имају број ставки већи од 1.

```
SELECT BrojRacuna, Count(*) as BrojStavki FROM stavkaracuna Group By
BrojRacuna HAVING Count(*) > 1
```

BrojRacuna	BrojStavki
1	3
2	2

Пример ArFun1: Аритметичке функције - Заокруглити продајну вредност ставки рачуна на 1 децималу.

```
SELECT RB, SifraProizvoda, Kolicina, ProdajnaCena, Round(ProdajnaVrednost,1) as
ProdajnaVrednost FROM stavkaracuna
```

RB	SifraProizvoda	Kolicina	ProdajnaCena	ProdajnaVrednost
1	pr1	5	45	225.0
2	pr2	1	45	45.0
3	pr3	2	56.34	112.7
1	pr1	7	12.56	87.9
2	pr2	7	12	84.0
1	pr1	3	45	135.0

Пример IfNull1: Показати како се користи функција ifnull.

Претпоставимо да треба да се унесе ставка за рачун број 3 за коју се не зна производ који је унет:

```
INSERT INTO StavkaRacuna VALUES ('3', '2', null, 3, 45.00, 0);
```

Претпоставимо да атрибут SifraProizvoda може да добије null vrednost.

Након извршења упита:

```
SELECT BrojRacuna, RB, SifraProizvoda, Kolicina, ProdajnaCena,
ProdajnaVrednost
FROM stavkaracuna
WHERE BrojRacuna = '3'
```

добија се:

BrojRacuna	RB	SifraProizvoda	Kolicina	ProdajnaCena	ProdajnaVrednost
3	1	pr1	3	45	135
3	2	null	4	67.45	0

Уколико желимо да код извештаја уместо null вредности ставимо нпр. вредност "nepoznata" писаћемо:

```
SELECT BrojRacuna, RB, Ifnull(SifraProizvoda, "nepoznata"),
Kolicina, ProdajnaCena,
ProdajnaVrednost FROM stavkaracuna
WHERE BrojRacuna = '3'
```

BrojRacuna	RB	SifraProizvoda	Kolicina	ProdajnaCena	ProdajnaVrednost
3	1	pr1	3	45	135
3	2	nepoznata	4	67.45	0

Пример ArFun2: Функција за рад са стринговима - За сваки рачун приказати број знакова назива партнера.

```
Select NazivPartnera, Length(NazivPartnera) as DuzinaNaziva From Racun
```

NazivPartnera	DuzinaNaziva
Meridian invest D.O.O	21
Perihard inzenjering	20
Sajam	5

Пример Ug1: Угњеждени упити (subquery) - Приказати рачуне код којих је продато више од 7 производа.

Пре него што се прикаже решење овога примера погледаћемо колико је по сваком рачуну продато производа:

```
Select BrojRacuna, Sum(Kolicina) as UkupnaKolicina FROM stavkaracuna Group
By BrojRacuna
```

BrojRacuna	UkupnaKolicina
1	8
2	14
3	7

Решење примера Ug1:

```
Select BrojRacuna, suma From (Select BrojRacuna, Sum(Kolicina) as suma From
stavkaracuna Group By BrojRacuna) as sume Where suma>7
```

BrojRacuna	suma
1	8
2	14

Пример Ug2: Угњеждени упити (subquery) - Приказати ставке рачуна које су направљене за Perihard inzenjering.

```
Select * From stavkaracuna Where BrojRacuna = (Select BrojRacuna from Racun
Where NazivPartnera = "Perihard inzenjering")
```

BrojRacuna	RB	SifraProizvoda	Kolicina	ProdajnaCena	ProdajnaVrednost
2	1	pr1	7	12.56	87.92
2	2	pr2	7	12	84

Правило: Унутрашњи упит мора да врати само једну вредност ако се у **Where** клаузули користи оператори поређења ($>=$, $<=$, $=$, $<$, $>$).

Уколико би направили још један рачун за *Perihard inzenjering*:

```
INSERT INTO Racun VALUES ('4', 'Perihard inženjering', 0, 'N', 'N');
са једном ставком:
```

```
INSERT INTO StavkaRacuna VALUES ('4', '1', 'pr1', 5, 71.68, 0);
```

тада би се при извршењу упита:

```
Select * From stavkaracuna Where BrojRacuna = (Select BrojRacuna from Racun
Where NazivPartnera = "Perihard inženjering")
```

јавила порука: *Subquery returns more than 1 row* (Подупит враћа више од 1 реда)

Тада се користи **IN** наредба уместо оператора =

```
Select * From stavkaracuna Where BrojRacuna IN (Select BrojRacuna from
Racun Where NazivPartnera = "Perihard inženjering")
```

BrojRacuna	RB	SifraProizvoda	Kolicina	ProdajnaCena	ProdajnaVrednost
2	1	pr1	7	12.56	87.92
2	2	pr2	7	12	84
4	1	pr1	5	71.68	0

Пример Ug3: Угњеждени упити (*subquery*) - Приказати рачун код кога је продато највише производа.

```
Select BrojRacuna, suma From (Select BrojRacuna, Sum(Kolicina) as suma From
stavkaracuna Group By BrojRacuna) as sume Where suma = (Select Max(suma)
From (Select BrojRacuna, sum(Kolicina) as suma From stavkaracuna Group By
BrojRacuna) as sume)
```

BrojRacuna	suma
2	14

Друго решење које је урађено преко погледа:

a) Прво се направи поглед

```
Create View Sume as (Select BrojRacuna, Sum(Kolicina) as suma From
stavkaracuna GroupBy BrojRacuna)
```

b) Над погледом се направи упит

```
Select BrojRacuna, suma From Sume Where suma = (Select Max(suma) From Sume)
```

Поглед се може преко *MySQL* SUBP-а запамтити.

Пример Uni1: Унија две табеле - Приказати све ставке рачуна 1 и 2 преко уније.

```
SELECT * FROM stavkaracuna WHERE BrojRacuna = '1'
```

UNION

```
SELECT * FROM stavkaracuna WHERE BrojRacuna = '2'
```

BrojRacuna	RB	SifraProizvoda	Kolicina	ProdajnaCena	ProdajnaVrednost
1	1	pr1	5	45	225
1	2	pr2	1	45	45
1	3	pr3	2	56.34	112.68
2	1	pr1	7	12.56	87.92
2	2	pr2	7	12	84

Пример Raz1: Разлика - Приказати рачуне свих пословних партнера осим *Perihard inženjeringa*.

```
SELECT * FROM racun WHERE NazivPartnera != "Perihard inženjering"
```

BrojRacuna	NazivPartnera	UkupnaVrednost	Obradjen	Storniran
1	Meridian invest D.O.O	0	N	N
3	Beogradski sajam	0	N	N

Пример Equijoin1: Спајање две или више табела - Спајање на једнакост (*equijoin*) - Пrikazati за сваку ставку рачуна име партнера за који је везана ставка.

```
SELECT nazivpartnera,stavkaracuna.* FROM racun,stavkaracuna WHERE  
racun.BrojRacuna = stavkaracuna.BrojRacuna
```

Nazivpartnera	BrojRacuna	RB	SifraProizvoda	Kolicina	ProdajnaCena	ProdajnaVrednost
Meridian invest D.O.O	1	1	pr1	5	45	225
Meridian invest D.O.O	1	2	pr2	1	45	45
Meridian invest D.O.O	1	3	pr3	2	56.34	112.68
Perihard inzenjerинг	2	1	pr1	7	12.56	87.92
Perihard inzenjerинг	2	2	pr2	7	12	84
Beogradski sajam	3	1	pr1	3	45	135
Beogradski sajam	3	2		4	67.45	0
Perihard inženjerинг	4	1	pr1	5	71.68	0

Пример CarJoin1: Спајање две или више табела - Декартов производ (*cartesian join*) - Пrikazati све комбинације слогова табела Racun i StavkaRacuna (из предходног примера смо изоставили Where клаузулу).

```
SELECT nazivpartnera,stavkaracuna.BrojRacuna,stavkaRacuna.RB FROM racun,  
stavkaracuna
```

nazivpartnera	BrojRacuna	RB
Meridian invest D.O.O	1	1
Perihard inzenjerинг	1	1
Beogradski sajam	1	1
Perihard inženjerинг	1	1
Meridian invest D.O.O	1	2
Perihard inzenjerинг	1	2
Beogradski sajam	1	2
Perihard inženjerинг	1	2
Meridian invest D.O.O	1	3
Perihard inzenjerинг	1	3
Beogradski sajam	1	3
Perihard inženjerинг	1	3
Meridian invest D.O.O	2	1
Perihard inzenjerинг	2	1
Beogradski sajam	2	1
Perihard inženjerинг	2	1
Meridian invest D.O.O	2	2
Perihard inzenjerинг	2	2
Beogradski sajam	2	2
Perihard inženjerинг	2	2
Meridian invest D.O.O	3	1
Perihard inzenjerинг	3	1
Beogradski sajam	3	1
Perihard inženjerинг	3	1
Meridian invest D.O.O	3	2
Perihard inzenjerинг	3	2
Beogradski sajam	3	2
Perihard inženjerинг	3	2
Meridian invest D.O.O	4	1
Perihard inzenjerинг	4	1
Beogradski sajam	4	1
Perihard inženjerинг	4	1

За 4 рачуна и 8 ставки рачуна има укупно 32 комбинације (парова) рачун-ставка рачуна.

Пример LeftJoin1: Спајање две или више табела - Спомоно спајање (*outer join*) – лево спајање (*left join*) - Приказати све шифре производа и ставке рачуна које су повезане са тим производима.

Креираћемо табелу производ:

```
CREATE TABLE `racun`.`proizvod` (
  `SifraProizvoda` VARCHAR(20) NOT NULL,
  PRIMARY KEY(`SifraProizvoda`)
)
```

Унећемо неколико производа.

```
INSERT INTO proizvod VALUES ('pr1');
INSERT INTO proizvod VALUES ('pr2');
INSERT INTO proizvod VALUES ('pr3');
INSERT INTO proizvod VALUES ('pr4');
INSERT INTO proizvod VALUES ('pr5');
```

Након тога ће се извршити упит:

```
SELECT Proizvod.SifraProizvoda, StavkaRacuna.BrojRacuna, StavkaRacuna.RB,
StavkaRacuna.SifraProizvoda
FROM Proizvod LEFT JOIN StavkaRacuna ON Proizvod.SifraProizvoda =
StavkaRacuna.SifraProizvoda;
```

SifraProizvoda	BrojRacuna	RB	SifraProizvoda
pr1	1	1	pr1
pr1	2	1	pr1
pr1	3	1	pr1
pr1	4	1	pr1
pr2	1	2	pr2
pr2	2	2	pr2
pr3	1	3	pr3
pr4			
pr5			

Правило: Приказују се сви слогови са леве (*LEFT*) стране од *JOIN* (*Proizvod*) и само они слогови са десне од *JOIN* (*StavkaRacuna*) који задовољавају услов *Proizvod.SifraProizvoda = StavkaRacuna.SifraProizvoda*

Пример RightJoin1: Спајање две или више табела - Спомоно спајање (*outer join*) – десно спајање (*right join*) - Приказати све ставке рачуна независно од тога да ли су везане за постојеће шифре производа

```
SELECT Proizvod.SifraProizvoda, StavkaRacuna.BrojRacuna, StavkaRacuna.RB
FROM Proizvod RIGHT JOIN StavkaRacuna ON Proizvod.SifraProizvoda =
StavkaRacuna.SifraProizvoda;
```

SifraProizvoda	BrojRacuna	RB
pr1	1	1
pr2	1	2
pr3	1	3
pr1	2	1
pr2	2	2
pr1	3	1
	3	2
pr1	4	1

Правило: Приказују се сви слогови са десне (*RIGHT*) стране од *JOIN* (*StavkaRacuna*) и само они слогови са леве стране од *JOIN* (*Racun*) који задовољавају услов *Proizvod.SifraProizvoda = StavkaRacuna.SifraProizvoda*

Пример SelfJoin1: Спајање две или више табела - Спомоно спајање (*outer join*) – Само спајање (*selfjoin*) - Приказати хијерархију производа почев од производа *pr1* који је на врху.

Проширићемо табелу производ са допунским атрибутом *Nadredjeni*.

```
ALTER TABLE `racun`.`proizvod` MODIFY COLUMN `SifraProizvoda` VARCHAR(20)
NOT NULL, ADD COLUMN `Nadredjeni` VARCHAR(20) NOT NULL;
```

Направићемо следећу хијерархијску саставницу (1 подређени производ улази и 1 надређени производ; у један надређени производ може да уђе више подређених производа). Производ *pr1* се састоји из производа *pr2* и *pr3*. Производ *pr2* се састоји од производа *pr4* и *pr5*.

```
UPDATE proizvod SET Nadredjeni = 'pr1' Where SifraProizvoda = 'pr2';
UPDATE proizvod SET Nadredjeni = 'pr1' Where SifraProizvoda = 'pr3';
UPDATE proizvod SET Nadredjeni = 'pr2' Where SifraProizvoda = 'pr4';
UPDATE proizvod SET Nadredjeni = 'pr2' Where SifraProizvoda = 'pr5';
UPDATE proizvod SET Nadredjeni = 'vrh' Where SifraProizvoda = 'pr1';
```

На крају ћемо извршити упит:

```
SELECT p.SifraProizvoda,p.Nadredjeni,s.SifraProizvoda as Podredjeni FROM
proizvod p, proizvod s Where p.SifraProizvoda = s.Nadredjeni
```

SifraProizvoda	Nadredjeni	Podredjeni
pr1	vrh	pr2
pr1	vrh	pr3
pr2	pr1	pr4
pr2	pr1	pr5

Објашњење: Табеле *p* и *s* се спајају преко *p.SifraProizvoda = s.Nadredjeni* на следећи начин:

Табела *p*

SifraProizvoda	Nadredjeni
pr1	vrh
pr2	pr1
pr3	pr1
pr4	pr2
pr5	pr2

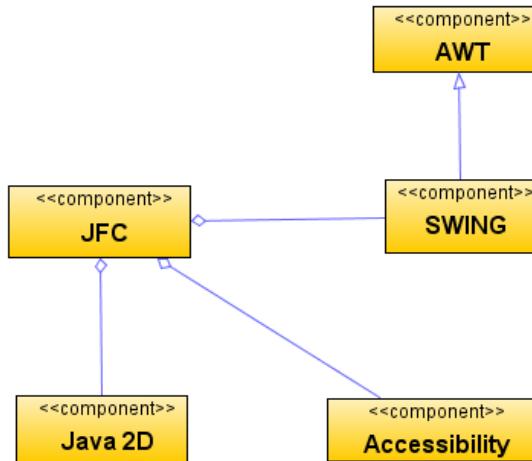
Табела *s*

SifraProizvoda	(ово је)	Nadredjeni
Podredjeni		
pr1		vrh
pr2		pr1
pr3		pr1
pr4		pr2
pr5		pr2

7. ГРАФИЧКИ КОРИСНИЧКИ ИНТЕРФЕЈС У ЈАВИ

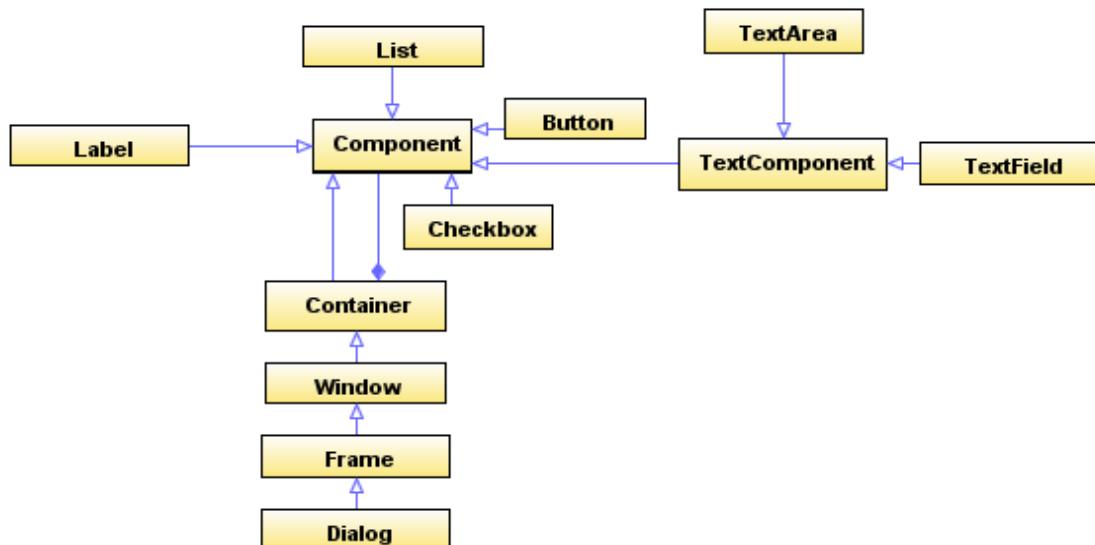
Програми који су засновани на графичком корисничком интерфејсу интеракцију са корисником остварују преко прозора (форми) на којима се налазе графичке компоненте за прихватавање и приказ података (текст поља, дугмићи, табеле...).

Java Fundation Classes (JFC) представља скуп класа које дају могућност креирања графичког корисничког интерфејса у Јави. Чине га неколико група класа (библиотека). На дијаграму компоненти приказане су неке од тих библиотека (*слика ГКИ1*).



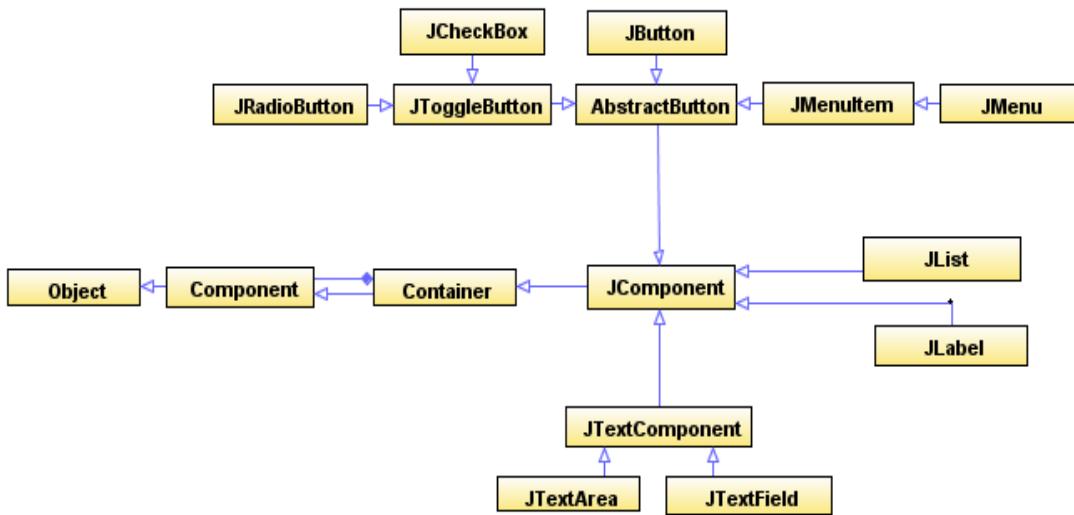
Слика ГКИ1: Дијаграм компоненти JFC библиотеке класа

Приликом реализације графичког корисничког интерфејса акценат ће бити на *Swing* компонентама. Оне представљају алтернативу *AWT* компонентама и имају велике предности у односу на њих. Имплементиране су у Јави и имају велики број заједничких интерфејса. Једна од предности *Swing* компоненти у односу на *AWT* компоненте је могућност измене изгледа графичких компоненти (оне имају *pluggable-look-and-feel*). Структура дела *AWT* компоненти приказана је дијаграмом класа (*слика ГКИ2*).



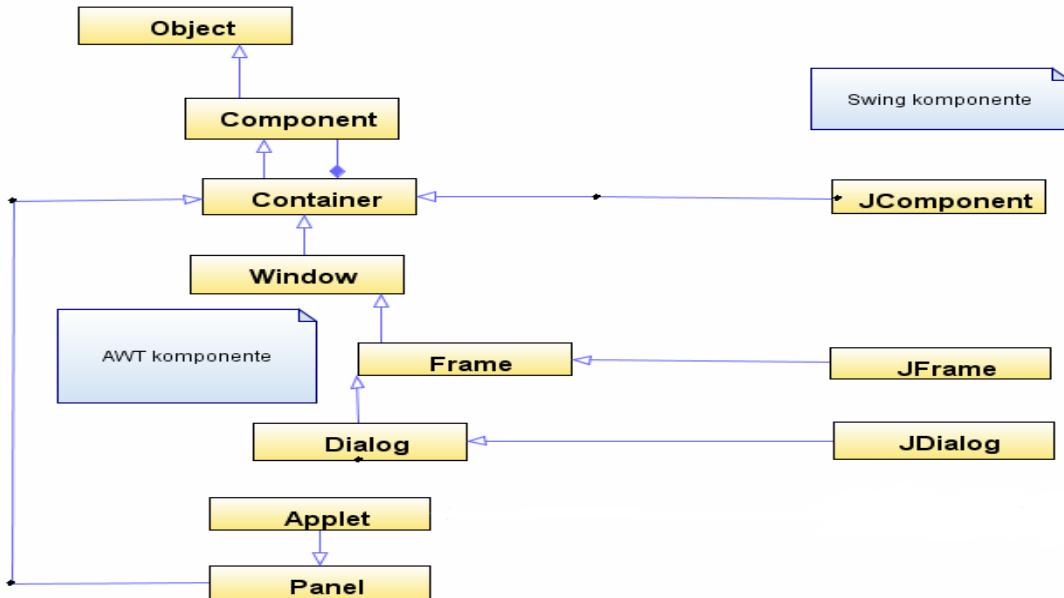
Слика ГКИ2: Неке од AWT компоненти приказане на дијаграму класа

Структура дела *Swing* компоненти приказана је дијаграмом класа (слика ГКИ3).



Слика ГКИ3: Неке од *Swing* компоненти приказане на дијаграму класа

Иако *Swing* компонентите представљају побољшање у односу на *AWT* компоненте, *AWT* компонентите имају значајно место у *JFC* библиотеци класа. Веза између *AWT* и *Swing* компоненти приказана је на дијаграму класа (слика ГКИ4).



Слика ГКИ4: Веза између *AWT* и *Swing* компоненти

На дијаграму класа (Слика ГКИ4) види се да неке *Swing* компоненте (на пример класа **JComponent** наслеђује *AWT* **Container** класу), док се на дијаграму класа на слици ГКИ3 се види да је **JComponent** класа родитељ класа за већину *Swing* компоненти).

7.1 Креирање графичких корисничких форми

У Јави се графичке корисничке форме могу направити на неколико начина:

- непосредним креирањем објекта класе `JFrame`, `JDialog` ...
- наслеђивањем класе `JFrame`, `JDialog` ...

Постављање графичких компоненти на екранску форму, панел или дијалог састоји се из неколико корака:

- креирање објекта графичке компоненте

```
btnPrikazi = new JButton();
```

- постављање вредности атрибута (*property*) за тај објекат

```
btnPrikazi.setText("Prikazi");
btnPrikazi.setBounds(1, 40, 100, 20);
```

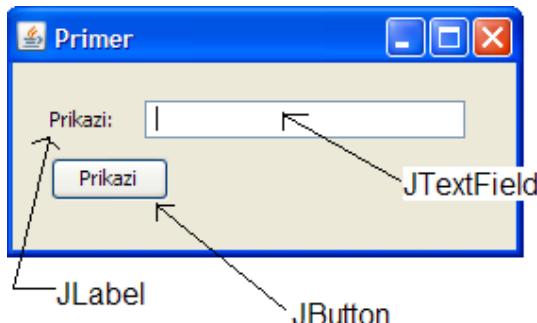
- постављање објекта графичке компоненте на екранску форму, панел или дијалог

```
Container kontejener = getContentPane();
kontejener.setLayout(null);
kontejener.add(btnPrikazi);
```

Пример ГКИ-1.

Програмски захтев:

Направити екранску форму као на слици (слика ГКИ5). Екранска форма садржи следеће компоненте: `JLabel` (Prikazi), `JTextField` (поље за унос) и `JButton` (Prikazi) као што је приказано на слици.



Слика ГКИ5: Изглед екранске форме

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (`Glavna.java`), док се у другој датотеци налази класа која представља екранску форму (`KForma.java`).

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class Glavna{
    public static void main(String[] args){
        KForma kf = new KForma("Ekrancka forma");
        kf.postaviKomponenteNaFormu();
        kf.otvoriformu();
    }
}
```

```

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Container;

class KForma extends JFrame{
    /* atributi klase koji predstavljaju graficke komponente */
    JButton btnPrikazi;
    JTextField txtPrikazi;
    JLabel labPrikazi;

    KForma(String naslovForme){
        super(naslovForme);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setBounds(200,200,300,100);
    }
    public void postaviKomponenteNaFormu(){
        /* kreirati odgovarajuce komponente */
        btnPrikazi = new JButton();
        txtPrikazi = new JTextField();
        labPrikazi = new JLabel();
        /* postaviti vrednosti atributa za objekte komponente */
        btnPrikazi.setText("Prikazi");
        labPrikazi.setText("Prikazi");
        /* odrediti pozicije komponenti na kontejneru forme */
        labPrikazi.setBounds(1,1,100,20);
        btnPrikazi.setBounds(1,40,100,20);
        txtPrikazi.setBounds(100,1,100,20);
        /* postaviti komponente na formu, odnosno kontejner forme */
        Container kontejener = getContentPane();
        kontejener.setLayout(null);
        kontejener.add(btnPrikazi);
        kontejener.add(txtPrikazi);
        kontejener.add(labPrikazi);
    }
    public void otvoriformu(){
        setVisible(true);
    }
}

```

Пример ГКИ-2.

Програмски захтев:

Направити екранску форму као на слици (слика ГКИ6). Екранска форма садржи следеће компоненте: *JLabel* (*Broj 1*, *Broj 2*, *Rezultat*), *JTextField* (поље за унос првог броја, поље за унос другог броја, поље за приказ резултата) и *JButton* (*Zbir* и *Razlika*) као што је приказано на слици.



Слика ГКИ6: Изглед екранске форме

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (*Glavna.java*), док се у другој датотеци налази класа која представља екранску форму (*KForma.java*).

```

/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

```

```

class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranska forma");
        kf.postaviKomponenteNaFormu();
        kf.otvoriformu();
    }
}

```

```

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Container;

class KForma extends JFrame{
    /* atributi klase koji predstavljaju graficke komponente */
    JLabel labBroj1;
    JLabel labBroj2;
    JLabel labRezultat;
    JTextField txtBroj1;
    JTextField txtBroj2;
    JTextField txtRezultat;
    JButton btnZbir;
    JButton btnRazlika;

    KForma(String naslovForme){
        super(naslovForme);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setBounds(200,200,350,110);
    }
    public void postaviKomponenteNaFormu(){
        /* kreirati odgovarajuce komponente */
        labBroj1 = new JLabel();
        labBroj2 = new JLabel();
        labRezultat = new JLabel();
        txtBroj1 = new JTextField();
        txtBroj2 = new JTextField();
        txtRezultat = new JTextField();
        btnZbir = new JButton();
        btnRazlika = new JButton();

        /* postaviti vrednosti atributa za objekte komponente */
        labBroj1.setText("Broj 1:");
        labBroj2.setText("Broj 2:");
        labRezultat.setText("Rezultat: ");
        btnZbir.setText("Zbir");
        btnRazlika.setText("Razlika");
        /* odrediti pozicije komponenti na kontejneru forme */
        labBroj1.setBounds(1,1,100,20);
        labBroj2.setBounds(1,25,100,20);
        labRezultat.setBounds(1,50,100,20);
        txtBroj1.setBounds(105,1,100,20);
        txtBroj2.setBounds(105,25,100,20);
        txtRezultat.setBounds(105,50,100,20);
        btnZbir.setBounds(210,1,100,20);
        btnRazlika.setBounds(210,25,100,20);

        /* postaviti komponente na formu, odnosno kontejner forme */
        Container kontejener = getContentPane();
        kontejener.setLayout(null);
        kontejener.add(labBroj1);
        kontejener.add(labBroj2);
        kontejener.add(labRezultat);
        kontejener.add(txtBroj1);
        kontejener.add(txtBroj2);
        kontejener.add(txtRezultat);
        kontejener.add(btnZbir);
        kontejener.add(btnRazlika);
    }
    public void otvoriformu(){
        setVisible(true);
    }
}

```

7.2 Управљање догађајима

Управљање догађајима на рачунарима, на највишем нивоу, врши оперативни систем. У једном тренутку у оперативном систему може бити активно неколико процеса (програма). Било који догађај који се деси над неким од програма (клика миша на компоненту на графичком корисничком интерфејсу или притисак типке на тастаури) региструје оперативни систем. Оперативни систем затим одређује који се програм извршава у простору где је регистрован догађај. Када утврди који се програм извршава тада оперативни систем том програму преноси тај догађај. Даљу контролу управљања догађаја врши конкретни програм у коме се десио догађај.

7.2.1 Концепт догађаја

Израда корисничког интерфејса захтева познавање **концепта догађаја и концепта слушалац догађаја**.

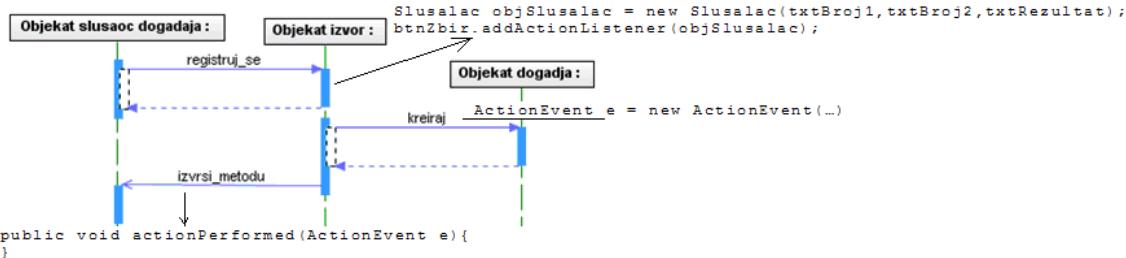
Догађај може да иницира нека од компонети графичког корисничког интерфејса (на пример дугме - JButton). Тада се за то дугме каже да оно представља *извор догађаја*.

Кликом миша на дугме креира се нови објекат који представља и идентификује тај догађај. То је *објекат догађаја* (у овом случају догађај клика миша креираће објекат класе ActionEvent). Објекат догађај садржи информације о извору догађаја и типу догађаја⁴¹.

Објекат догађај се прослеђује до објекта *слушаоца догађаја* (енг. *listener*). Прослеђивање објекта догађај (у примеру објекта класе ActionEvent) представља позив методе у објекту слушаоца догађаја.

Да би се објекат догађај проследио до објекта слушаоца неопходно је да се повеже извор догађаја и слушалац дугме. То се ради регистровањем објекта слушаоца у објекту извора догађаја.

Дефинисање догађаја у Јави приказано је на дијаграму секвенци (слика ГКИ7).



Слика ГКИ7: Дијаграм секвенци дефинисања догађаја

7.2.2 Класе догађаја

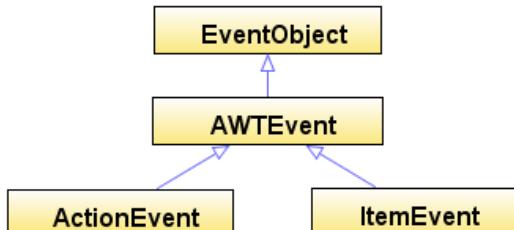
У Јави постоје две категорије догађаја: догађаји ниског нивоа и семантички догађаји.

7.2.1.1 Класе семантичких догађаја

Извор ових догађаја јесу Јавине компоненте графичког корисничког интерфејса (*Swing* и *AWT* компоненте) и објекат догађај настаје као резултат акције ових компоненти.

Два типа семантичких догађаја се најчешће користе при креирању графичког корисничког интерфејса: *ActionEvent* и *ItemEvent*. Хијерархија ових класа дата је дијаграмом класа (слика ГКИ8).

⁴¹ Једна компонента може бити извор различитих типова догађаја



Слика ГКИ8: Дијаграм класа семантичкx догађаја

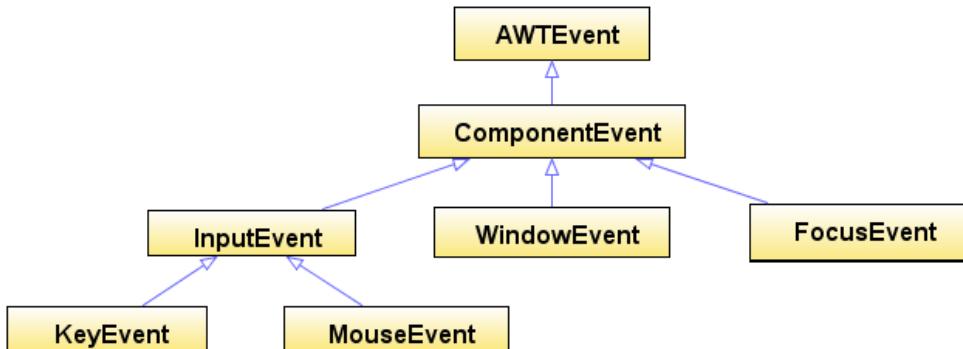
ActionEvent – креира објекат извор када изврши неку акцију

ItemEvent – креира објекат извор када се изабери или поништи избор (понуђена вредност) у објекту извору

7.2.1.2 Класе догађаја ниског нивоа

Извор ових догађаја јесу Јавине компоненте, а настају као резултат акција миша и/или тастатуре над њима. Посебну врсту ових догађаја креирају класе екранске форме(JFrame, JDialog...) приликом промене величине . Пример једног таквог догађаја јесте притисак типке на тастатури над компонентом JTextField.

Четири типа догађаја ниског ниво се најчешће користе при креирању графичког корисничког интерфејса: FocusEvent, MouseEvent, KeyEvent и WindowsEvent. Хијерархија ових класа дата је дијаграмом класа (слика ГКИ9)



Слика ГКИ9: Дијаграм класа догађаја ниског ниво

FocusEvent – креира објекат извор када тај објекат добије или изгуби фокус

MouseEvent – креира објекат извор када се на тај објекат изврши нека акција мишем (на пример клик миша или превлачење миша преко компоненте)

KeyEvent – креира објекат извор када се над тим објектом изврши акција притиска типке на тастатури

WindowsEvent – креира објекат извор (објекти класе Window и њених подкласа) када се изврши акција смањења објекта, повећање објекта извора ...

7.2.2 Слушаоци догађаја

Класа слушалац догађаја јесте Јавина класа која имплементира интерфејс слушаоца догађаја. Сви интерфејси слушалаца догађаја наслеђују интерфејс java.util.EventListener. Сваки слушалац догађаја прихвата одговарајући објекат догађај и дефинише методе за реаговање на тај догађај.

7.2.2.1 Слушаоци семантичких догађаја

Јава API дефинише неколико интерфејса слушаоца семантичких догађаја од којих се најчешће користе ActionListener и ItemListener. Ови интерфејси имају по једну методу. Тако ActionListener има методу:

- public void actionPerformed (ActionEvent e)

док ItemListener садржи методу:

- public void itemStateChanged (ItemEvent e)

Извор ових догађаја могу бити различите Јавине swing компонете и то:

- дугмаћи (JButton, JCheckBox...)
- менији (JMenuItem, JMenu...)
- текст компоненте (JTextField...)
- комбо поља

за догађаје ActionEvent, док за догађаје ItemEvent извори могу бити swing компонете и то:

- дугмаћи
- менији
- комбо поља

Пример ГКИ-3.

Програмски захтев:

Направити екранску форму као на слици (слика ГКИ10). Дефинисати догађај на дугме zbir и то на следећи начин: притиском на дугме zbir у текст пољу rezultat приказати збир бројева из текст поља broj 1 и текст поља broj 2. Такође дефинисати догађај на дугме razlika и то на следећи начин: притиском на дугме razlika у текст пољу rezultat приказати разлику бројева из текст поља broj 1 и текст поља broj 2.



Слика ГКИ10: Изглед екранске форме

Програм се састоји од три Јава датотеке. У првој Јава датотеци налази се главни програм (Glavna.java). У другој Јава датотеци налази се имплементација слушалац класе (Slusalac.java), док се у трећој датотеци налази имплементација класе која представља екранску форму (KForma.java).

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class Glavna{
    public static void main(String[] args){
        KForma kf = new KForma("Ekranska forma");
        kf.postaviKomponenteNaFormu();
        kf.postaviSlusaoce();
        kf.otvoriformu();
    }
}
```

```

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

class Slusalac implements ActionListener{
    public void actionPerformed(ActionEvent e){
    }
}

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Container;
class KForma extends JFrame{
    ...
    KForma(String naslovForme){
        /* programski kod je identican kao u konstruktoru za primer KI-2 */
        ...
    }
    public void postaviKomponenteNaFormu(){
        /* programski kod je identican kao u metodi u primeru KI-2 */
        ...
    }
    public void otvoriFormu(){
        /* programski kod je identican kao u metodi u primeru KI-2 */
        ...
    }
    public void postaviSlusaocu(){
        /* postaviti slusaoce na dugme btnZbir i dugme btnRazlika */
        Slusalac objSlusalac = new Slusalac();
        btnZbir.addActionListener(objSlusalac);
        btnRazlika.addActionListener(objSlusalac);
    }
}

```

У овом примеру дате су три класе. За разлику од примера *Пример-ГКИ2* у овом примеру додата је класа `Slusalac`. Ова класа представља слушалац класу како за дугме збир тако и за дугме разлику. Два проблема у овом примеру треба да се реше:

- метода `actionPerformed` у класи `Slusalac` мора да зна која је компонента извор догађаја

У методи `public void actionPerformed(ActionEvent e)` параметар `e` (објекат класе `ActionEvent`) чува информацију о извору догађаја

```

public void actionPerformed(ActionEvent e){
    if( ((JButton)e.getSource()).getText().toString().equals("Zbir") ){
        System.out.println("Zbir");
    }

    if( ((JButton)e.getSource()).getText().toString().equals("Razlika") ){
        System.out.println("Razlika");
    }
}

```

- метода `actionPerformed` у класи `Slusalac` мора да има референцу на компоненте из класе `KForma`

Приликом креирања објекта класе `Slusalac` конструктору се прослеђују референце на компоненте екранске форме.

```

class Slusalac implements ActionListener{
    JTextField txtBroj1;
    JTextField txtBroj2;
    JTextField txtRezultat;

    public Slusalac(JTextField txtBroj1,JTextField txtBroj2, JTextField
                    txtRezultat){
        this.txtBroj1 = txtBroj1;
        this.txtBroj2 = txtBroj2;
        this.txtRezultat = txtRezultat;
    }
    public void actionPerformed(ActionEvent ex){
        ...
    }
}

```

Ниже је дат комплетан програмски код за све три класе.

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranска форма");
        kf.postaviKomponenteNaFormu();
        kf.postaviSlusaoce();
        kf.otvoriformu();
    }
}
```

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JButton;
import javax.swing.JTextField;
class Slusalac implements ActionListener{

    JTextField txtBroj1;
    JTextField txtBroj2;
    JTextField txtRezultat;

    public Slusalac(JTextField txtBroj1,JTextField txtBroj2,JTextField txtRezultat){
        this.txtBroj1 = txtBroj1;
        this.txtBroj2 = txtBroj2;
        this.txtRezultat = txtRezultat;
    }
    public void actionPerformed(ActionEvent e){
        try{
            int b1 = Integer.parseInt (txtBroj1.getText().trim());
            int b2 = Integer.parseInt (txtBroj2.getText().trim());
            int rezultat=0;
            if( ((JButton)e.getSource()).getText().toString().equals("Zbir") ){
                rezultat = b1+b2;
            }

            if( ((JButton)e.getSource()).getText().toString().equals("Razlika") ){
                rezultat = b1-b2;
            }

            txtRezultat.setText(String.valueOf(rezultat));
        }catch(Exception ex){
        }
    }
}
```

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Container;

class KForma extends JFrame{

    /* atributi klase koji predstavljaju graficke komponente */
    JLabel labBroj1;
    JLabel labBroj2;
    JLabel labRezultat;
    JTextField txtBroj1;
    JTextField txtBroj2;
    JTextField txtRezultat;
    JButton btnZbir;
    JButton btnRazlika;

    KForma(String naslovForme){
        super(naslovForme);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setBounds(200,200,350,110);
    }
}
```

```

public void postaviKomponenteNaFormu(){
    /* kreirati odgovarajuce komponente */
    labBroj1 = new JLabel();
    labBroj2 = new JLabel();
    labRezultat = new JLabel();
    txtBroj1 = new JTextField();
    txtBroj2 = new JTextField();
    txtRezultat = new JTextField();
    btnZbir = new JButton();
    btnRazlika = new JButton();
    /* postaviti vrednosti atributa za objekte komponente */
    labBroj1.setText("Broj 1:");
    labBroj2.setText("Broj 2:");
    labRezultat.setText("Rezultat: ");
    btnZbir.setText("Zbir");
    btnRazlika.setText("Razlika");
    /* odrediti pozicije komponenti na kontejneru forme */
    labBroj1.setBounds(1,1,100,20);
    labBroj2.setBounds(1,25,100,20);
    labRezultat.setBounds(1,50,100,20);
    txtBroj1.setBounds(105,1,100,20);
    txtBroj2.setBounds(105,25,100,20);
    txtRezultat.setBounds(105,50,100,20);
    btnZbir.setBounds(210,1,100,20);
    btnRazlika.setBounds(210,25,100,20);
    /* postaviti komponente na formu, odnosno kontejner forme */
    Container kontejener = getContentPane();
    kontejener.setLayout(null);
    kontejener.add(labBroj1);
    kontejener.add(labBroj2);
    kontejener.add(labRezultat);
    kontejener.add(txtBroj1);
    kontejener.add(txtBroj2);
    kontejener.add(txtRezultat);
    kontejener.add(btnZbir);
    kontejener.add(btnRazlika);
}
public void otvoriFormu(){
    setVisible(true);
}
public void postaviSlusaoce(){
    /* postaviti slusaoce na dugme btnZbir i dugme btnRazlika */
    Slusalac objSlusalac = new Slusalac(txtBroj1,txtBroj2,txtRezultat);
    btnZbir.addActionListener(objSlusalac);
    btnRazlika.addActionListener(objSlusalac);
}
}

```

Регистровање слушалаца догађаја могуће је коришћењем анонимних класа. За сваки догађај креира се анонимна Јава класа. Анонимне класе се креирају као унутрашње класе (*inner class*). Креирање анонимних класа скраћује писање кода у Јави.

```

class KForma extends JFrame{
    KForma(String naslovForme){
        ...
    }
    public void postaviKomponenteNaFormu(){
        ...
    }
    public void otvoriFormu(){
        ...
    }
    public void postaviSlusaoce(){
        /* postaviti slusaoce na dugme btnZbir i dugme btnRazlika */
        btnZbir.addActionListener(new ActionListener(){
            /* ovo je anonimna klasa koja implementira metode iz interfejsa */
            public void actionPerformed(ActionEvent e){
                try{
                    int b1 = Integer.parseInt (txtBroj1.getText().trim());
                    int b2 = Integer.parseInt (txtBroj2.getText().trim());
                    int rezultat=0;
                    rezultat = b1+b2;
                    txtRezultat.setText(String.valueOf(rezultat));
                }catch(Exception ex){
                }
            }
        });
    }
}

```

```

        btnRazlika.addActionListener(new ActionListener(){
            /* ovo je anonimna klasa koja implementira metode iz interfejsa */
            public void actionPerformed(ActionEvent e){
                try{
                    int b1 = Integer.parseInt (txtBroj1.getText().trim());
                    int b2 = Integer.parseInt (txtBroj2.getText().trim());
                    int rezultat=0;
                    rezultat = b1-b2;
                    txtRezultat.setText(String.valueOf(rezultat));
                }catch(Exception ex){
                }
            }
        });
    }
}

```

7.2.2.2 Слушаоци догађаја ниског нивоа

Јава API дефинише велики број интерфејса слушаоца догађаја ниског нивоа (WindowListener, MouseListener, KeyListener, FocusListener...)

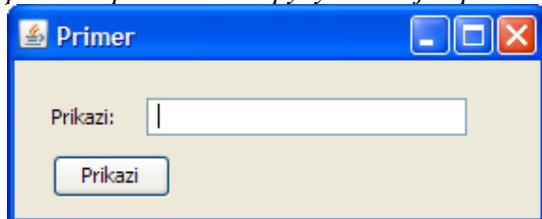
Интерфејс MouseListener дефинише следеће методе:

- public void mouseEntered (MouseEvent e) – позива се када миш уђе у област објекта извора догађаја
- public void mouseExited (MouseEvent e) – позива се када миш изађе из области објекта извора догађаја
- public void mousePressed (MouseEvent e) – позива се када се притисне на објекат извора догађаја
- public void mouseReleased (MouseEvent e) – позива се када се дугме миша отпusti са објекта извора догађаја
- public void mouseClicked (MouseEvent e) – позива се када се дугме миша кликне на објекат извора догађаја (притисне и отпusti)

Пример ГКИ5.

Програмски захтев:

Направити екранску форму као на слици (слика ГКИ11). Дефинисати догађај клика миша на дугме prikazi на следећи начин: притиском на дугме prikazi у текст пољу prikazi приказати поруку: Мии је притиснут!



Слика ГКИ11: Изглед екранске форме

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (Glavna.java), док се у другој датотеци налази класа која представља екранску форму (KForma.java).

```

/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

```

```

class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranska forma");
        kf.postaviKomponenteNaFormu();
        kf.postaviSlusaoce();
        kf.otvoriFormu();
    }
}

```

```

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Container;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
/*
    klasa KForma predstavlja i klasu forma i klasu slusalac dogadjaja
    posto implementira interfejs slusaoca
*/

class KForma extends JFrame implements MouseListener {
    /* atributi klase koji predstavljaju graficke komponente */
    JButton btnPrikazi;
    JTextField txtPrikazi;
    JLabel labPrikazi;

    KForma(String naslovForme) {
        ...
    }
    public void postaviKomponenteNaFormu() {
        ...
    }
    public void otvoriFormu() {
        ...
    }

    public void postaviSlusaoce() {
        /* postavi slusaoca dogadjaja na dugme */
        btnPrikazi.addMouseListener(this);
    }

    /* metode interfejsa MouseListener */
    public void mouseClicked(MouseEvent e) {
        txtPrikazi.setText("Mis je pritisnut!");
    }

    public void mousePressed(MouseEvent e) {
    }

    public void mouseReleased(MouseEvent e) {
    }

    public void mouseEntered(MouseEvent e) {
    }

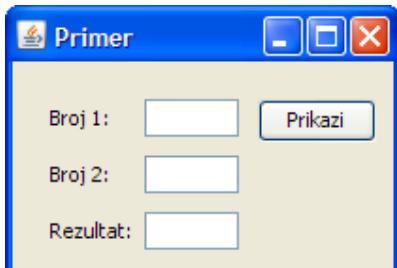
    public void mouseExited(MouseEvent e) {
    }
}

```

Пример ГКИ-6.

Програмски захтев:

Направити екранску форму као на слици (слика ГКИ12). Дефинисати догађај на дугме prikazi на следећи начин: притиском на дугме prikazi у текст пољу rezultat приказати збир бројева из текст поља broj 1 и текст поља broj 2, а када се дугме пусти у пољу rezultat приказати разлику бројева.



Слика ГКИ12: Изглед екранске форме

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (`Glavna.java`), док се у другој датотеци налази класа која представља екранску формуу (`KForma.java`).

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranска форма");
        kf.postaviKomponenteNaFormu();
        kf.postaviSlusaoce();
        kf.otvoriFormu();
    }
}
```

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Container;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
class KForma extends JFrame implements MouseListener{
    /* atributi klase koji predstavljaju graficke komponente */
    JLabel labBroj1;
    JLabel labBroj2;
    JLabel labRezultat;
    JTextField txtBroj1;
    JTextField txtBroj2;
    JTextField txtRezultat;
    JButton btnPrikazi;

    KForma(String naslovForme){
        ...
    }
    public void postaviKomponenteNaFormu(){
        ...
    }
    public void otvoriFormu(){
        ...
    }
    public void postaviSlusaoce(){
        /* postavi slusaoca dogadjaja na dugme */
        btnPrikazi.addMouseListener(this);
    }

    /* metode interfejsa MouseListener */
    public void mouseClicked(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
        try{
            int a = Integer.parseInt(txtBroj1.getText());
            int b = Integer.parseInt(txtBroj2.getText());
            txtRezultat.setText(String.valueOf(a+b));
        }catch(Exception ex){
        }
    }
}
```

```

public void mouseReleased(MouseEvent e) {
    try{
        int a = Integer.parseInt(txtBroj1.getText());
        int b = Integer.parseInt(txtBroj2.getText());
        txtRezultat.setText(String.valueOf(a-b));
    }catch(Exception ex){
    }
}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}

}

```

Пример ГКИ -7.*Програмски захтев:*

Направити екранску форму као на слици (Слика ГКИ13). Дефинисати догађај на дугме zbir на следећи начин: притиском на дугме zbir у текст пољу rezultat приказати збир бројева из текст поља broj 1 и текст поља broj 2. Такође дефинисати догађај на дугме razlika на следећи начин: притиском на дугме razlika у текст пољу rezultat приказати разлику бројева из текст поља broj 1 и текст поља broj 2.



Слика ГКИ13: Изглед екранске форме

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (`Glavna.java`), док се у другој датотеци налази класа која представља екранску форму (`KForma.java`).

```

/*
 * @author Dusan Savic
 * SILAB - Laboratoriya za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranska forma");
        kf.postaviKomponenteNaFormu();
        kf.postaviSlusaoce();
        kf.otvoriformu();
    }
}

```

```

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Container;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
class KForma extends JFrame{
    /* atributi klase koji predstavljaju graficke komponente */
    JLabel labBroj1;
    JLabel labBroj2;
    JLabel labRezultat;
    JTextField txtBroj1;
    JTextField txtBroj2;
    JTextField txtRezultat;
    JButton btnZbir;
    JButton btnRazlika;
}

```

```

KForma(String naslovForme) {
    ...
}
public void postaviKomponenteNaFormu() {
    ...
}
public void otvoriFormu() {
    ...
}
public void postaviSlusaoce(){
    /* slusalac na dugme zbir */
    btnZbir.addMouseListener(new MouseListener(){
        public void mouseClicked(MouseEvent e) {
            try{
                int a = Integer.parseInt(txtBroj1.getText());
                int b = Integer.parseInt(txtBroj2.getText());
                txtRezultat.setText(String.valueOf(a+b));
            }catch(Exception ex){
            }
        }
        public void mousePressed(MouseEvent e) {
        }
        public void mouseReleased(MouseEvent e) {
        }
        public void mouseEntered(MouseEvent e) {
        }
        public void mouseExited(MouseEvent e) {
        }
    });
    /* slusalac na dugme razlika */
    btnRazlika.addMouseListener(new MouseListener(){
        public void mouseClicked(MouseEvent e) {
            try{
                int a = Integer.parseInt(txtBroj1.getText());
                int b = Integer.parseInt(txtBroj2.getText());
                txtRezultat.setText(String.valueOf(a-b));
            }catch(Exception ex){
            }
        }
        public void mousePressed(MouseEvent e) {
        }
        public void mouseReleased(MouseEvent e) {
        }
        public void mouseEntered(MouseEvent e) {
        }
        public void mouseExited(MouseEvent e) {
        }
    });
}
}

```

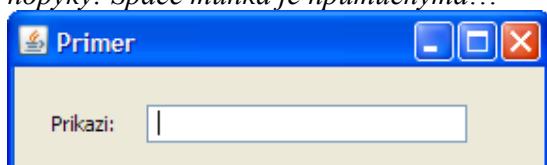
Интерфејс `KeyListener` дефинише следеће методе:

- public void keyPressed (KeyEvent e)
 - public void keyReleased (KeyEvent e)
 - public void keyTyped (KeyEvent e)

Пример ГКИ-8.

Програмски захтев:

Направити екранску форму као на слици (слика ГКИ-14). Дефинисати догађај притиска тинке *enter* и тинке *space* на тастатури над текстуалним пољем *prikazi* на следећи начин: притиском на тинку *enter* у текстуално поље приказати поруку: *Enter тинка је притиснута!!!*, а притиском на тинку *space* у текстуално поље приказати поруку: *Space тинка је притиснута!!!*



Слика ГКИ14: Изглед екранске форме

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (`Glavna.java`), док се у другој датотеци налази класа која представља екранску форму (`KForma.java`).

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranska forma");
        kf.otvoriformu();
    }
}
```

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Container;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;

class KForma extends JFrame{
    /* atributi klase koji predstavljaju graficke komponente */
    JTextField txtPrikazi;
    JLabel labPrikazi;

    KForma(String naslovForme){
        super(naslovForme);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setBounds(200,200,500,60);

        /* postavi komponente na formu */
        postaviKomponenteNaFormu();
        /* postavi slusaocu dogadjaja */
        postaviSlusaoce();
    }

    private void postaviKomponenteNaFormu(){
        ...
    }

    private void postaviSlusaoce(){
        txtPrikazi.addKeyListener(new KeyListener(){
            public void keyPressed (KeyEvent e){
                int tipka = e.getKeyCode();
                if (tipka == KeyEvent.VK_ENTER) {
                    txtPrikazi.setText("Enter tipka je pritisnuta");
                }
                if (tipka == KeyEvent.VK_SPACE) {
                    txtPrikazi.setText("Space tipka je pritisnuta");
                }
            }
            public void keyReleased (KeyEvent e){
            }
            public void keyTyped (KeyEvent e){
            }
        });
    }

    public void otvoriformu(){
        setVisible(true);
    }
}
```

7.3 Адаптер класе

При дефинисању слушаоца семантичких догађаја потребно је имплементирати одговарајући интерфејс слушаоца.

Пошто неки од тих интерфејса имају више од једне методе тада је у класи слушаоцу која имплементира интерфејс слушаоца потребно извршити прекривање свих метода из интерфејса слушаоца⁴². На пример: Дефинисањем догађаја клика миша прекрива се само метода `public void mouseClicked(MouseEvent e)` из интерфејса `MouseListener`. Због тога су у Јави уведене адаптер класе које имплементирају интерфејсе слушалаца и чије методе немају дефинисано понашање (тела ових метода су празна).

На тај начин дефинисање класе ослушкивача могуће је и наслеђивањем адаптер класе. За сваки интерфејс ослушкивача постоји одговарајућа адаптер класа (*Слика ГКИ15*).

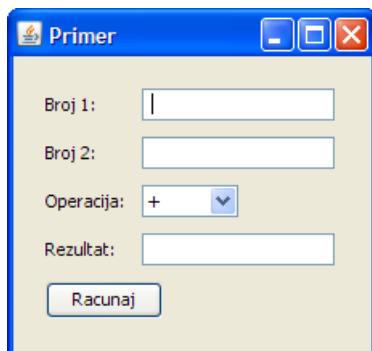
Интерфејс ослушкивача	Адаптер класа
WindowListener	WindowAdapter
MouseListener	MouseAdapter
KeyListener	KeyAdapter
FocusListener	FocusAdapter

Слика 15: Адаптер класе за дефинисање догађаја ниског нивоа

Пример ГКИ-9.

Програмски захтев:

Направити екранску форму као на слици (*Слика ГКИ16*). У текст поља (број 1 и број 2) корисник уноси бројеве, док из комбо бокса корисник бира операцију (+,-). Кликом на дугме *Racunaj* приказати резултат операције над унетим операндима (broj1 i broj 2) у текст поље *rezultat*.



Слика ГКИ16: Изглед екранске форме

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (`Glavna.java`), док се у другој датотеци налази класа која представља екранску форму (`KForma.java`).

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranska forma");
        kf.otvoriformu();
    }
}
```

⁴² некада постоји потреба за прекривањем само једне методе из интерфејса слушаоца.

```

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JComboBox;
import java.awt.eventMouseListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

class KForma extends JFrame{
    /* atributi klase koji predstavljaju graficke komponente */
    private JButton btnRacunaj;
    private JComboBox cboxOperacija;
    private JLabel labBroj1;
    private JLabel labBroj2;
    private JLabel labOperacija;
    private JLabel labRezultat;
    private JTextField txtBroj1;
    private JTextField txtBroj2;
    private JTextField txtRezultat;

    KForma(String naslovForme){
        super(naslovForme);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setBounds(200,200,350,300);

        /* postavi komponente na formu */
        postaviKomponenteNaFormu();
        /* postavi vrednosti u kombo polje operacije */
        postaviOperacije();
        /* postavi slusaocu */
        postaviSlusaoce();
    }

    private void postaviKomponenteNaFormu(){
        ...
    }
    private void postaviOperacije(){
        cboxOperacija.addItem(new String("+"));
        cboxOperacija.addItem(new String("-"));
    }
    private void postaviSlusaoce(){
        btnRacunaj.addMouseListener(new MouseAdapter(){
            public void mouseClicked(MouseEvent e){
                racunaj(e);
            }
        });
    }
    private void racunaj(MouseEvent e){
        try{
            String operacija = cboxOperacija.getSelectedItem().toString();
            int a = Integer.parseInt(txtBroj1.getText().trim());
            int b = Integer.parseInt(txtBroj2.getText().trim());
            if (operacija.equals("+")) saberi(a, b);
            if (operacija.equals("-")) oduzmi(a, b);
        }catch(Exception ex){
        }
    }
    private void saberi(int a, int b){
        txtRezultat.setText(String.valueOf(a+b));
    }
    private void oduzmi(int a,int b){
        txtRezultat.setText(String.valueOf(a-b));
    }
    public void otvoriFormu(){
        setVisible(true);
    }
}

```

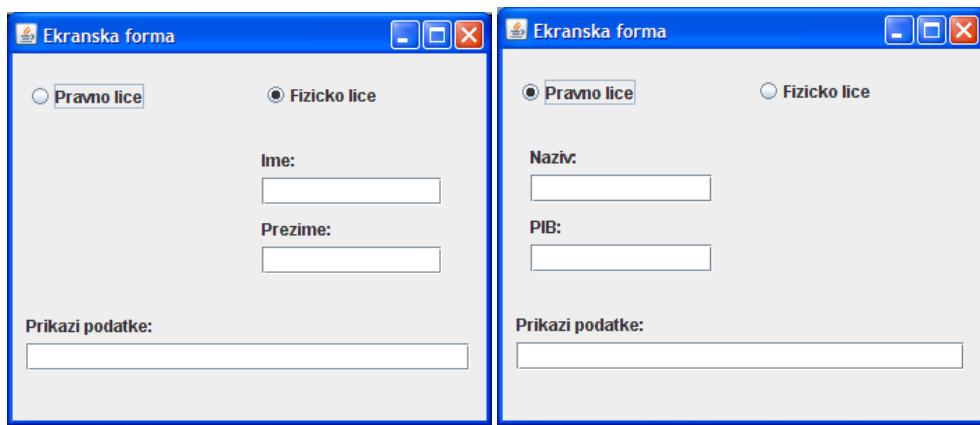
7.4 Рад са компонентом JPanel

Компонента *JPanel* представља контејнер компоненту. Контејнер компоненте су Јавине компоненте (класе) које могу да садрже друге компоненте.

Пример ГКИ-10.

Програмски захтев:

Направити екранску форму као на слици (слика ГКИ17). Програм треба да омогућити кориснику да изабере да ли жељи да унесе податке за правно или физичко лице. Уколико уноси податке за правно лице корисник уноси назив и пиб, док приликом уноса података за физичко лице уноси име и презиме. Када корисник притисне тику enter а налази се у текстуалном пољу за приказ података тада се у том текстуалном пољу приказује порука о правном или физичком лицу у зависности од тога које је лице изабрано.



Слика ГКИ17: Изглед екранске форме

У овом примеру компоненте у којима се уносе подаци за правно лице постављају се на један панел, док се компоненте у којима се уносе подаци за физичко лице постављају на други панел. Након покретања програма изабрана је опција за унос физичког лица чиме је и панел који садржи компоненте за унос физичког лица видљив. Уколико корисник промени избор и селектује правно лице тада панел за унос података за физичко лице постаје скрижен, а панел са компонентама за унос правног лица постаје видљив.

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (*Glavna.java*), док се у другој датотеци налази класа која представља екранску форму (*KForma.java*).

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranska forma");
        kf.otvoriformu();
    }
}
```

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JPanel;
import java.awt.Container;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.ButtonGroup;
import javax.swing.JRadioButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

class KForma extends JFrame{
    /* atributi klase koji predstavljaju graficke komponente */
    private JRadioButton izbFizickoLice;
    private JRadioButton izbPravnoLice;
    private ButtonGroup izbor;
    private JLabel labIme;
    private JLabel labNaziv;
    private JLabel labPIB;
    private JLabel labPrezime;
    private JLabel labPrikaz;
    private JPanel panelFL;
    private JPanel panelPL;
    private JTextField txtIme;
    private JTextField txtNaziv;
    private JTextField txtPIB;
    private JTextField txtPrezime;
    private JTextField txtPrikaz;

    KForma(String naslovForme){
        super(naslovForme);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setBounds(200,200,350,300);
        /* postavi komponente na formu */
        postaviKomponenteNaFormu();
        /* selektuj izbor fizicko lice */
        postaviPocetniPanel(true);
        /* postavi slusaoce */
        postaviSlusaoce();

    }
    private void postaviKomponenteNaFormu(){
        /* kreiranje objekata komponenti */
        izbor = new javax.swing.ButtonGroup();
        izbPravnoLice = new javax.swing.JRadioButton();
        panelFL = new javax.swing.JPanel();
        labIme = new javax.swing.JLabel();
        txtIme = new javax.swing.JTextField();
        labPrezime = new javax.swing.JLabel();
        txtPrezime = new javax.swing.JTextField();
        panelPL = new javax.swing.JPanel();
        labNaziv = new javax.swing.JLabel();
        txtNaziv = new javax.swing.JTextField();
        labPIB = new javax.swing.JLabel();
        txtPIB = new javax.swing.JTextField();
        izbFizickoLice = new javax.swing.JRadioButton();
        labPrikaz = new javax.swing.JLabel();
        txtPrikaz = new javax.swing.JTextField();

        getContentPane().setLayout(null);
        panelFL.setLayout(null);
        panelPL.setLayout(null);

        /* izbor pravno lice */
        izbPravnoLice.setText("Pravno lice");
        izbPravnoLice.setBounds(10, 20, 93, 23);
        getContentPane().add(izbPravnoLice);
        izbor.add(izbPravnoLice);

        /* izbor fizicko lice */
        izbFizickoLice.setText("Fizicko lice");
        izbFizickoLice.setBounds(180, 20, 93, 20);
        getContentPane().add(izbFizickoLice);
```

```

izbor.add(izbFizickoLice);

/* labela ime */
labIme.setText("Ime:");
labIme.setBounds(10, 10, 120, 14);
panelFL.add(labIme);
/* tekst polje ime */
txtIme.setBounds(10, 30, 130, 20);
panelFL.add(txtIme);

/* labela prezime */
labPrezime.setText("Prezime:");
labPrezime.setBounds(10, 60, 130, 14);
panelFL.add(labPrezime);
/* tekst polje prezime */
txtPrezime.setBounds(10, 80, 130, 20);
panelFL.add(txtPrezime);

/* labela naziv */
labNaziv.setText("Naziv:");
labNaziv.setBounds(10, 10, 120, 14);
panelPL.add(labNaziv);
/* tekstualno polje naziv */
txtNaziv.setBounds(10, 30, 130, 20);
panelPL.add(txtNaziv);

/* labela PIB */
labPIB.setText("PIB:");
labPIB.setBounds(10, 60, 130, 14);
panelPL.add(labPIB);
/* tekstualno polje PIB */
txtPIB.setBounds(10, 80, 130, 20);
panelPL.add(txtPIB);

/* paneli */
panelFL.setBounds(170, 60, 150, 120);
panelPL.setBounds(10, 60, 150, 120);
getContentPane().add(panelFL);
getContentPane().add(panelPL);

/* prikaz podataka */
labPrikaz.setText("Prikazi podatke:");
labPrikaz.setBounds(10, 190, 150, 14);
getContentPane().add(labPrikaz);
txtPrikaz.setBounds(10, 210, 320, 20);
getContentPane().add(txtPrikaz);

}

private void postaviPocetniPanel(boolean signal){
    izbFizickoLice.setSelected(signal);
    izbPravnoLice.setSelected(!signal);
    panelFL.setVisible(signal);
    panelPL.setVisible(!signal);
}

private void postaviSlusaoce(){
    izbPravnoLice.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            if (izbPravnoLice.isSelected()){
                postaviPocetniPanel(false);
                txtPrikaz.setText("");
            }
        }
    });
    izbFizickoLice.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            if (izbFizickoLice.isSelected()){
                postaviPocetniPanel(true);
                txtPrikaz.setText("");
            }
        }
    });
}

/* definisanje dogadjaja pritisak tipke uz pomoc adapter klase KeyAdapter*/
txtPrikaz.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent e) {
        pritisakTipke(e);
    }
});
}

```

```

private void pritisakTipke(KeyEvent e){
    String tekstPoruka="";

    int tipka = e.getKeyCode();
    if (tipka == KeyEvent.VK_ENTER) {
        if (izbFizickoLice.isSelected()){
            /* fizicko lice */
            String ime = txtIme.getText().trim();
            String prezime = txtPrezime.getText().trim();
            tekstPoruka = ime + ", "+prezime;
        }
        if (izbPravnoLice.isSelected()){
            /* pravno lice */
            String naziv = txtNaziv.getText().trim();
            String pib = txtPIB.getText().trim();
            tekstPoruka = naziv + ", "+pib;
        }
        txtPrikaz.setText(tekstPoruka);
    }
}

public void otvoriFormu(){
    setVisible(true);
}
}

```

7.5 Рад са дефинисаним дијалозима

Статичке методе које се налазе у класи JOptionPane пружају могућност лаког креирања неколико типова модалних дијалога и то: дијалога за приказ порука (метода showMessageDialog), дијалог са питањем (метода showConfirmDialog), дијалог за унос података (метода showInputDialog), дијалог са избором понуђених опција (showOptionDialog).

Пример ГКИ-11.

Програмски захтев:

*Направити екранску форму као на слици (слика ГКИ18). Дефинисати догађај над дугметом прикази на следећи начин: кликом на дугме приказати дијалог са поруком *Ovo dugme niste smeli da kliknete!!!**



Слика ГКИ18: Изглед екранске форме

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (Glavna.java), док се у другој датотеци налази класа која представља екранску форму (KForma.java).

```

/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekrancka forma");
        kf.otvoriFormu();
    }
}

```

```

import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JOptionPane;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Container;

class KForma extends JFrame {

    /* atributi klase koji predstavljaju graficke komponente */
    JButton btnPrikazi;

    KForma(String naslovForme){
        super(naslovForme);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setBounds(200,200,300,100);
        /* postavi komponente na formu */
        postaviKomponenteNaFormu();
        /* postavi slosaoce */
        postaviSlusaoce();
    }

    private void postaviKomponenteNaFormu(){
        ...
    }
    private void postaviSlusaoce(){
        btnPrikazi.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                otvoriDialog(e);
            }
        });
    }
    private void otvoriDialog(ActionEvent e){
        String poruka = "Ovo dugme niste smeli da kliknete!!!";
        String naslov = "Poruka";
        JOptionPane.showMessageDialog(this,poruka ,naslov, JOptionPane.ERROR_MESSAGE);
    }
    public void otvoriFormu(){
        setVisible(true);
    }
}

```

Пример ГКИ-12

Програмски захтев:

Направити екранску форму као на слици (слика ГКИ19) . Дефинисати догађај над дугметом прикажи на следећи начин: кликом на дугме прикажи приказати дијалог са питањем *Da li zelite da napustite aplikaciju?* Уколико корисник одговори потврдно прекинути извршење програма у супротном затворити дијалог.



Слика ГКИ19: Изглед екранске форме

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (*Glavna.java*), док се у другој датотеци налази класа која представља екранску форму (*KForma.java*).

```

/*
 * @author Dusan Savic
 * SILAB - Laboratoriya za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

```

```
class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranska forma");
        kf.otvoriFormu();
    }
}
```

```
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JOptionPane;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Container;

class KForma extends JFrame {

    /* atributi klase koji predstavljaju graficke komponente */
    JButton btnPrikazi;

    KForma(String naslovForme){
        super(naslovForme);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setBounds(200,200,300,100);
        /* postavi komponente na formu */
        postaviKomponenteNaFormu();
        /* postavi slosaoce */
        postaviSlusaoca();
    }
    private void postaviKomponenteNaFormu(){
        ...
    }
    private void postaviSlusaoca(){
        btnPrikazi.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                otvoriDialog(e);
            }
        });
    }
    private void otvoriDialog(ActionEvent e){
        int rezultat = JOptionPane.showConfirmDialog(this,"Da li zelite da
                                               napustite aplikaciju","Poruka",JOptionPane.YES_NO_OPTION );
        /* 0 = da */
        if (rezultat == 0) System.exit(0);
    }
    public void otvoriFormu(){
        setVisible(true);
    }
}
```

Пример ГКИ-13.

Програмски захтев:

*Направити екранску форму као на слици (слика ГКИ20). Дефинисати догађај над дугметом прикази на следећи начин: кликом на дугме прикази приказати дијалог са питањем *Da li zelite da napustite aplikaciju?* Уколико корисник унутар дијалога унесе вредноса *D* прекинути извршење програма у супротном затворити дијалог.*



Слика ГКИ20: Изглед екранске форме

Програм се састоји од две Јава датотеке. У првој Јава датотеци налази се главни програм (*Glavna.java*), док се у другој датотеци налази класа која представља екранску форму (*KForma.java*).

```

/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranska forma");
        kf.otvoriformu();
    }
}

```

```

import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JOptionPane;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Container;

class KForma extends JFrame {
    /* atributi klase koji predstavljaju graficke komponente */
    JButton btnPrikazi;
    KForma(String naslovForme){
        super(naslovForme);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setBounds(200,200,300,100);
        /* postavi komponente na formu */
        postaviKomponenteNaFormu();
        /* postavi slosaoce */
        postaviSlusaoce();
    }
    private void postaviKomponenteNaFormu(){
        /* kreirati odgovarajuce komponente */
        btnPrikazi = new JButton();
        /* postaviti vrednosti atributa za objekte komponente */
        btnPrikazi.setText("Prikazi");
        /* odrediti pozicije komponenti na kontejneru forme */
        btnPrikazi.setBounds(50,20,100,20);
        /* postaviti komponente na formu, odnosno kontejner forme */
        Container kontejener = getContentPane();
        kontejener.setLayout(null);
        kontejener.add(btnPrikazi);
    }
    private void postaviSlusaoce(){
        btnPrikazi.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                otvoridialog(e);
            }
        });
    }
    private void otvoridialog(ActionEvent e){
        String pitanje = "Da li zelite da napustite aplikaciju (D/N)?";
        int tip = JOptionPane.QUESTION_MESSAGE;
        String rezultat = JOptionPane.showInputDialog(this,pitanje , "Unesite vrednost",tip);
        if (rezultat.equalsIgnoreCase("d")) System.exit(0);
    }
    public void otvoriformu(){
        setVisible(true);
    }
}

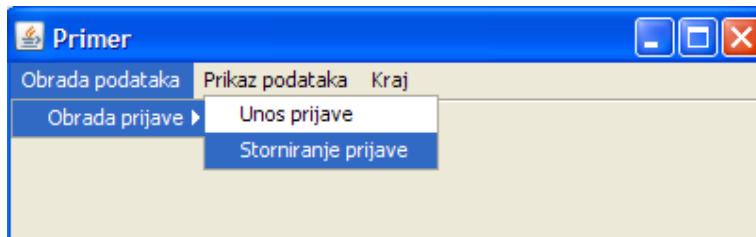
```

7.6 Рад са менијима

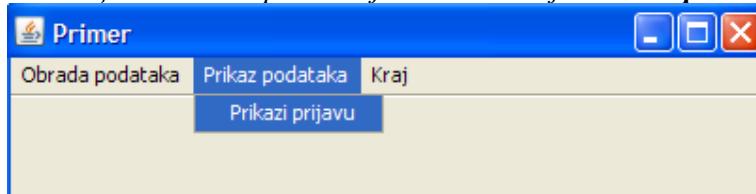
Приликом креирања менија у Јава апликацији користе се следеће класе: `javax.swing.JMenuBar`, `javax.swing.JMenu`, `javax.swing.JMenuItem`. Начин коришћења ових класа објаснићемо на примеру који следи.

Пример ГКИ-14.

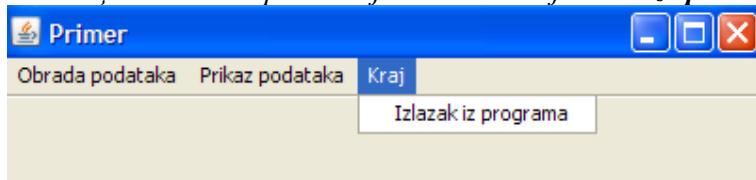
Написати Јава програм којим ће се креирати екранска форма као на сликама ГКИ21.1, ГКИ21.2 и ГКИ21.3. Уколико корисник из менија изабере Крај -> Излазак из програме. Уколико корисник из менија изабере Приказ података -> Пrikажи пријаву, приказати кориснику дијалог са поруком: Нема унетих пријава.



На слици ГКИ21.1 приказан је изглед менија **Obrada podataka**.

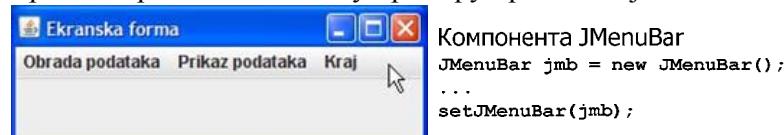


На слици ГКИ21.2 приказан је изглед менија **Prikaza podataka**.

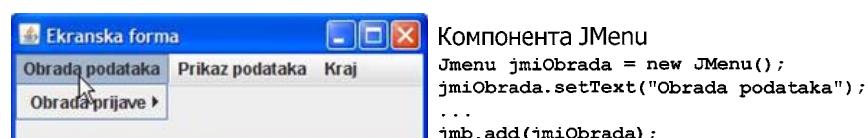


На слици ГКИ21.3 приказан је изглед менија **Kraj**.

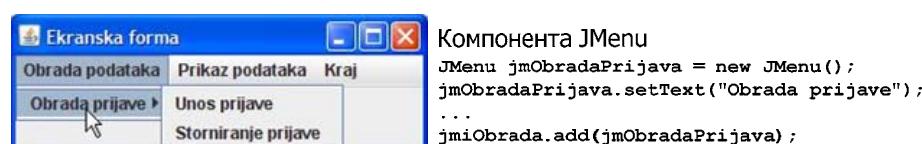
Приказ коришћених класа у примеру приказано је ниже на слици.



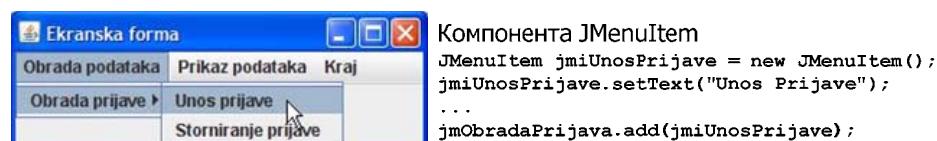
Компонента JMenuBar
JMenuBar jmb = new JMenuBar();
...
setJMenuBar(jmb);



Компонента JMenu
Jmenu jmiObrada = new JMenu();
jmiObrada.setText("Obrada podataka");
...
jmb.add(jmiObrada);



Компонента JMenuItem
JMenuItem jmiUnosPrijave = new JMenuItem();
jmiUnosPrijave.setText("Unos Prijava");
...
jmObrada.add(jmiUnosPrijave);



Компонента JMenuItem
JMenuItem jmiUnosPrijave = new JMenuItem();
jmiUnosPrijave.setText("Unos Prijava");
...
jmObrada.add(jmiUnosPrijave);

Ниже је дат комплетан програмски код.

```
/*
 * @author Dusan Savic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
```

```

class Glavna{
    public static void main(String[]args){
        KForma kf = new KForma("Ekranska forma");
        kf.otvoriformu();
    }
}

import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JOptionPane;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Container;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;

class KForma extends JFrame {
    /* atributi klase koji predstavljaju graficke komponente */
    /* meni Kraj*/
    private JMenu jmKraj;
    /* meni Obrada podatka -> Obrada prijave */
    private JMenu jmObradaPrijava;
    /* meni Prikaz podataka */
    private JMenu jmPrikaz;
    /* meni Obrada podatka */
    private JMenu jmiObrada;
    /* menu bar je komponenta koja se nalazi na vrhu forme */
    /* na koju se postavljaju meniji */
    private JMenuBar jmb;

    private JMenuItem jmiIzlazIzPrograma;
    private JMenuItem jmiPrikaziPrijave;
    private JMenuItem jmiStorniranjePrijave;
    private JMenuItem jmiUnosPrijave;

    KForma(String naslovForme){
        super(naslovForme);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setBounds(200,200,300,100);
        /* postavi komponente na formu */
        postaviKomponenteNaFormu();
        /* postavi slosaoce */
        postaviSlusaocce();
    }
    private void postaviKomponenteNaFormu(){
        /* kreirati odgovarajuce komponente */
        jmb = new javax.swing.JMenuBar();
        jmiObrada = new javax.swing.JMenu();
        jmObradaPrijava = new javax.swing.JMenu();
        jmiUnosPrijave = new javax.swing.JMenuItem();
        jmiStorniranjePrijave = new javax.swing.JMenuItem();
        jmPrikaz = new javax.swing.JMenu();
        jmiPrikaziPrijave = new javax.swing.JMenuItem();
        jmKraj = new javax.swing.JMenu();
        jmiIzlazIzPrograma = new javax.swing.JMenuItem();

        jmiObrada.setText("Obrada podataka");
        jmObradaPrijava.setText("Obrada prijave");
        jmiUnosPrijave.setText("Unos prijave");
        jmiStorniranjePrijave.setText("Storniranje prijave");
        jmPrikaz.setText("Prikaz podataka");
        jmiPrikaziPrijave.setText("Prikazi prijavu");
        jmKraj.setText("Kraj");
        jmiIzlazIzPrograma.setText("Izlazak iz programa");

        jmObradaPrijava.add(jmiUnosPrijave);
        jmObradaPrijava.add(jmiStorniranjePrijave);
        jmiObrada.add(jmObradaPrijava);
        jmb.add(jmiObrada);
        jmPrikaz.add(jmiPrikaziPrijave);
        jmb.add(jmPrikaz);
        jmKraj.add(jmiIzlazIzPrograma);
        jmb.add(jmKraj);
    }
}

```

```

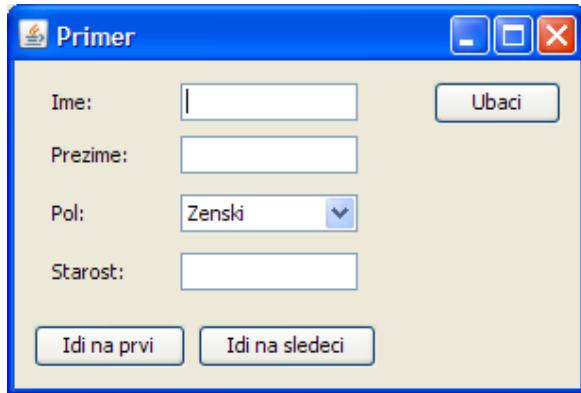
/* postaviMenuBar na vrh forme */
setJMenuBar(jmb);
}
private void postaviSlusaocce() {
    jmiIzlazIzPrograma.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e){
            System.exit(0);
        }
    });
    jmiPrikaziPrijavu.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e){
            prikaziDijalog();
        }
    });
}

public void prikaziDijalog(){
    String poruka = "Nema unetih prijava";
    JOptionPane.showMessageDialog(this,poruka,"Poruka",JOptionPane.INFORMATION_MESSAGE);
}
public void otvoriformu(){
    setVisible(true);
}
}

```

Задаци :**ЗГКИ1:** Направити екранску форму која треба да омогући кориснику да :

- унесе податке о особи (име,презиме, пол и старост)
- податке о особи сачува у неку од Јавиних колекција Јавину колекцију
- прегледа колекцију (подаци за текућу особу приказују се у текстуалним пољима за унос)

Предлог екранске форме дат је на слици *ГКИ22*:

Слика ГКИ22. Предлог екранске форме

ЗГКИ2: Направити екранску форму која треба да омогући кориснику да:

- унесе бројеве и сачува их у колекцију
- омогући кориснику да пронађе:
 - најмањи број
 - суму парних елемената
 - број елемената који су већи од аритметичке средине парних ел.колекције

Предлог екранске форме дат је на слици *ГКИ23*:



Слика ГКИ23. Предлог екранске форме

Питања:

1. Објаснити разлику између *Swing* и *AWT* компоненти?
2. За које класе у Јави се каже да су контејнер класе? Дати пример.
3. Објаснити концепт догађаја у Јави.
4. Објаснити разлику између семантичких догађаја и догађаја ниског нивоа.
5. За које класе у Јави се каже да су класе слушаоци догађаја?
6. На који начин се дефинишу слушаоци догађаја.

8. МЕХАНИЗАМ РЕФЛЕКСИЈЕ У ЈАВИ

Деф. Рефлексија је механизам који омогућава добијање основних информација о класи и свим њеним чланицама. Ове информације називамо метаподацима, које се у случају објектно орјентисаног програмирања, чувају у мета-објектима.⁴³

У време извршавања програма све учитане класе имају објекат класе `Class`, у којима се налазе мета подаци.

8.1 Добијање основних информација о класи коришћењем Class објекта

Једна од најчешће коришћених класа из пакета `java.lang.reflect` је класа `Class`. Класа `Class` има методе које се користе за испитивање класа. Мета-објекат садржи све информације о класи коју представља. Коришћењем метода ове класе можемо добити име класе, списак свих метода и атрибута класе. Интересантно је да и примитивни типови, и `void` имају одговарајуће `.class` мета-објекте⁴⁴.

Пример Реф1: Креирати класу `X`. Приказати основне информације о класи `X` коришћењем механизма рефлексије.

```
package refl;

/**
 * @author Vojislav Stanojevic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
public class Refl {

    /** Creates a new instance of Refl */
    public Refl() {
    }

    public static void main(String[] args) {
        Class metaX = X.class;
        System.out.println("Основне информације о класи X:" + metaX);
    }
}

class X{
    int a;
}
```

8.1.2 Испитивање атрибута класе коришћењем рефлексије

Механизам рефлексије, поред већ наведених ствари омогућава рад са атрибутима класе. Класа `Class` поред наведених метода има и методе за преглед и рад са атрибутима класе. У табели 2.1 приказане су методе класе `Class` са њиховим кратким описом.

Табела Реф1 методе класе `Class` за рад са атрибутима.

Метода	Опис методе
<code>Field getField(String name)</code>	Враћа објекат класе <code>Field</code> који представља атрибут класе, који има исти назив као онај који је прослеђен као параметар методе. Ова метода узима у обзир јавне атрибуте класе, као и јавне атрибуте које је класа наследила.
<code>Field[] getFields()</code>	Враћа низ објекат класе <code>Field</code> који представља све јавне атрибуте класе, као и јавне атрибуте које је класа наследила.

⁴³ Мета-објекат је објекат који описује класу и њене елементе

⁴⁴ На пример `int.class` представља мета-објекат простог типа инт

<pre>Field getDeclaredField(String name)</pre>	Враћа објеката класе Field који представља атрибут класе, који има исти назив као онај који је прослеђен као параметар методе. Ова метода узима у обзир само атрибуте декларисане у оквиру класе, не и оне које је класа наследила. Метода не разматра право приступа до атрибута.
<pre>Field[] getDeclaredFields()</pre>	Враћа низ објекат класе Field који представља све атрибуте који су декларисане у оквиру класе, не и наслеђене. Метода не разматра право приступа до атрибута.

Пример Реф2. Дефинисати класу X која има атрибут atr1 целобројног типа и атрибут atr2 типа String. Коришћењем механизма рефлексије, приказати назив класе и атрибуте које класа има.

```
package ref2;

import java.lang.reflect.Field;

/**
 * @author Vojislav Stanojevic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
public class Ref2 {

    /** Creates a new instance of Ref2 */
    public Ref2() {
    }

    public static void main(String[] args) {
        Class metaX = X.class;
        System.out.println("informacije o klasi X");
        System.out.println("*****");
        System.out.println("naziv klase:" + metaX.getName());
        Field [] atributi = metaX.getDeclaredFields();
        System.out.println("*****");
        System.out.println("atributi klase:");
        for (int i = 0; i < atributi.length; i++) {
            System.out.println(atributi[i]);
        }
        System.out.println("*****");
    }
}

class X{
    int atr1;
    String atr2;
}

//Rezultat:
//informacije o klasi X
//*****
//naziv klase:ref2.X
//*****
//atributi klase:
//int ref2.X.atr1
//java.lang.String ref2.X.atr2
//*****
```

8.1.3. Испитивање метода у времену извршавања

Класа Class поседује и методе које омогућавају преглед и рад са методама класе коју препрезентују. Тако коришћењем ових метода можемо врло лако да сазнамо списак свих метода које има нека класа. Основне методе класе Class које се користе за испитивање (introspection) метода класе приказане су у табели 2.

Tabela Реф2. метода класе Class за испитивање метода.

Метода	Опис методе
Method getMethod (String name, Class[] parameterTypes)	Враћа објекат класе Method који представља јавну методу (било декларисану било наслеђену) датог објекта, са параметрима који одговарају другом параметру који је прослеђен овој методи
Method[] getMethods ()	Враћа низ објеката класе Method који представља све јавне методе репрезентоване класе (било декларисане било наслеђене)
Method getDeclaredMethod (String name, Class[] parameterTypes)	Враћа објекат класе Method који представља декларисану методу (не укључује наслеђене методе) датог објекта, са параметрима који одговарају другом параметру који је прослеђен овој методи.
Method[] getDeclaredMethods ()	Враћа низ објеката класе Method који представља све декларисане методе репрезентоване класе (не само јавне).

Пример Реф3. Дефинисати класу X која има атрибуте atr1 целобројног типа и atr2 типа String, и методе m1() , и m2() која враћа void а као параметар има String . Коришћењем механизма рефлексије, приказати назив класе и све методе ове класа.

```
package ref3;

import java.lang.reflect.Method;

/**
 * @author Vojislav Stanojevic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
public class Ref3 {

    /** Creates a new instance of Ref3 */
    public Ref3() {
    }

    public static void main(String[] args) {
        Class metaX = X.class;
        System.out.println("informacije o klasi X");
        System.out.println("*****");
        System.out.println("naziv klase:"+metaX.getName());
        Method [] metode = metaX.getDeclaredMethods();
        System.out.println("*****");
        System.out.println("metode klase:");
        for (int i = 0; i < metode.length; i++) {
            System.out.println(metode[i]);
        }
        System.out.println("*****");
    }

    class X{
        int atr1;
        String atr2;

        void m1(){
            System.out.println("Ovo je metoda m1!");
        }

        void m2(String s){
            System.out.println("Ovoj metodi je prosledjen parametar "+s);
        }
    }

    //Rezultat:
    //informacije o klasi X
    //*****
    //naziv klase:ref3.X
    //*****
    //metode klase:
    //void ref3.X.m1()
    //void ref3.X.m2(java.lang.String)
    //*****
}
```

8.2. Коришћење Field класе за рад са атрибутима објекта

Информације о самим атрибутима налазе се у објектима класе `Field`. Видели смо које су методе класе `Class` којим можемо добити појављивања класе `Field` које представљају одговарајуће атрибуте. У табели 3.2 Дат је преглед метода класе `Field` са кратким описом тих метода

Tabela Рef3. Методе класе `Field` за рад са атрибутима.

Метода	Опис методе
<code>Class getType()</code>	Враћа објекат класе <code>Class</code> који представља декларисани тип податка за атрибут који је представљен датим објектом.
<code>Class getDeclaringClass()</code>	Враћа објекат класе <code>Class</code> који представља класу у којој је декларисан атрибут који је представљен овим <code>Field</code> објектом
<code>String getName()</code>	Враћа назив атрибута који је представљен датим <code>Field</code> објектом
<code>int getModifiers()</code>	Враћа модификатор атрибута, који је представљена датим <code>Field</code> објектом, трансформисан у <code>int</code>
<code>Object get(Object obj)</code>	Враћа објекат који представља вредност атрибута (представљеног датим <code>Field</code> објектом) објекта <code>obj</code>
<code>boolean getBoolean(Object obj)</code>	Враћа <code>boolean</code> који представља вредност атрибута (представљеног датим <code>Field</code> објектом) објекта <code>obj</code> ,
.....	Постоји аналогна метода, за враћање вредности атрибута за сваки прости тип у јави
<code>Object set(Object obj, Object vred)</code>	Поставља вредност параметра <code>vred</code> атрибуту објекта <code>obj</code> , представљеног датим <code>Field</code> објектом
<code>void setInt(Object obj, int vred)</code>	Поставља вредност параметра <code>vred</code> (типа <code>int</code>), атрибуту објекта <code>obj</code> , представљеног датим <code>Field</code> објектом
.....	Постоји аналогна метода, за постављање вредности атрибута за сваки прости тип у Јави

8.2.1 Постављање и узимање вредности атрибута

Постављање и узимање вредности атрибута, као што се могло видети, ради се помоћу одговарајућих `get` и `set` метода, представљених у претходној табели.

За све `set` методе важи, да уколико приступ до датог атрибута није дозвољен(што проверава Java Language Access Control) тада се дешава `IllegalAccessException` изузетак, а ако је тип вредност која је послата као параметар методе не одговара типу атрибута који представља дати `Field` објекат десиће се `IllegalArgumentException` изузетак.

Пример Рef4. Дефинисати класу `Osoba`, која има атрибуте `jmbg` мана `String`, `imePrezime` мана `String` и `godineStarosti` целобројног типа. Приказати назив класе и све атрибуте. У приказ укључити тип атрибута.

```
package ref4;

import java.lang.reflect.Field;

/**
 * @author Vojislav Stanojevic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 *http://silab.fon.bg.ac.yu
 */

public class Ref4 {

    /** Creates a new instance of Ref4 */
    public Ref4() {
    }
    public static void main(String[] args) {
        Class metaX = Osoba.class;
        System.out.println("informacije o klasi Osoba");
        System.out.println("*****");
        System.out.println("naziv klase:"+metaX.getName());
        Field [] atributi = metaX.getDeclaredFields();
```

```

System.out.println("*****");
System.out.println("atributi klase:");
for (int i = 0; i < atributi.length; i++) {
    System.out.println("atribut "+atributi[i].getName()+" tipa
"+atributi[i].getType().getName());
}
System.out.println("*****");
}

class Osoba {

    String jmbg;
    String imePrezime;
    int godineStarosti;

    /** Creates a new instance of Osoba */
    public Osoba() {
    }
}

//Rezultat:
//informacije o klasi Osoba
//*****
//naziv klase:ref4.Osoba
//*****
//atributi klase:
//atribut jmbg tipa java.lang.String
//atribut imePrezime tipa java.lang.String
//atribut godineStarosti tipa int
//*****

```

8.2.2 Испитивање и рад са модификаторима

Код рада са Field и Method класама помињани су модификатори који стоје уз одговарајући чланицу класе. Класе Field и Method реализују интерфејс Member, тако да свака од њих имплементира методе Class getDeclaringClass(), String getName() као и методу int getModifiers(). За ово поглавље посебно је интересантна метода int getModifiers() која враћа целобројну вредност која представља трансформисани модификатор.

У Јави постоји 11 модификатора:

public	static	native
volatile	protected	abstract
Synchronized	strctfp	private
final	transient	

Сваком од ових модификатора је додељен одговарајући бит у оквиру int вредности која је враћена getModifiers методом. У Јавином АПИ-ју за рефлексију постоји Modifier класа која врши декодирање враћеног модификатора. У табели 2.3.1 је дат преглед метода који врше декодирање модификатора.

Табела Реф4. Методе класе Field за рад са атрибутима.

Метода	Опис методе
static Boolean isPublic(int mod)	Враћа true ако и само ако public модификатор постоји међу модификаторима који су представљени параметром mod који је типа int
static boolean isFinal(int mod)	Враћа true ако и само ако final модификатор постоји међу модификаторима који су представљени параметром mod који је типа int
.....	Аналогно претходним методама постоје методе за сваки од 11 типова модификатора

8.2.3 Приступ не-јавним чланицама класе

Већ је раније напоменуто да се приликом приступа сваке од чланица класе врши провера права приступа до те чланице класе. Уколико није дозвољен приступ некој од чланица класе, коришћењем рефлексије ипак је могуће приступити истој коришћењем метода класе `java.lang.reflect.AccessibleObject`. Обе класе и `Field` и `Method` наслеђују поменуту класу тако да обе имају методу `void setAccessible(boolean flag)` којом може да се стопира, односно поново активира провера права приступа до одређене чланице класе.

```
Clanica_Klase.setAccessible(true);
```

Поред методе `void setAccessible(boolean flag)`, ова класа има методу `boolean isAccessible()` која враћа `true` вредност уколико је могуће приступити чланици класе или `false` уколико то није могуће. Постоји метода `static void setAccessible(AccessibleObject[] nizClanica, boolean flag)` која низу чланица класа поставља право приступа у зависности од параметра `flag`.

Пример Реф5. Дефинисати класу `Osoba`, која има атрибуте `jmbg` тима `String`, `imePrezime` тима `String` и `godineStarosti` тима `int`. Креирати објекат класе `Osoba`, коришћењем механизма рефлексије поставити вредности атрибута а затим приказати податке о особи.

```
package ref5;

import java.lang.reflect.Field;

/**
 * @author Vojislav Stanojevic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
public class Ref5 {

    /** Creates a new instance of Ref5 */
    public Ref5() {
    }

    public static void main(String[] args) {
        Osoba o = new Osoba();
        // ovo je jos jedan nacin da se dobije Class objekat koji predstavlja klasu Osoba
        Class metaOsoba = o.getClass();
        try {

            Field atrJmbg = metaOsoba.getDeclaredField("jmbg");
            // metoda getField("jmbg") bacila bi izuzetak NoSuchMethodException
            // zato sto je nacin pristupa do atributa jmbg private
            atrJmbg.setAccessible(true);
            atrJmbg.set(o,new String("0707987711811"));
            Field atrImeP = metaOsoba.getDeclaredField("imePrezime");
            atrImeP.set(o,new String("Pera Peric"));
            System.out.println("Podaci o Osobi:");
            Field [] atributi= metaOsoba.getDeclaredFields();
            for (int i = 0; i < atributi.length; i++) {
                if (!atributi[i].isAccessible()) {atributi[i].setAccessible(true);}
                System.out.println(atributi[i].getName()+"="+atributi[i].get(o));
            }

        } catch (SecurityException ex) {
            ex.printStackTrace();
        } catch (IllegalArgumentException ex) {
            ex.printStackTrace();
        } catch (NoSuchFieldException ex) {
            ex.printStackTrace();
        }
        catch (IllegalAccessException ex) {
            ex.printStackTrace();
        }
    }
}
```

```

}

class Osoba {

    private String jmbg;
    public String imePrezime;
    public int godineStarosti;

    /** Creates a new instance of Osoba */
    public Osoba() {
    }
}

//Rezultat:
//Podaci o Osobi:
//jmbg=0707987711811
//imePrezime=Pera Peric
//godineStarosti=0

```

8.3. Коришћење Method класе за рад са методама

У претходном поглављу дат је преглед метода класе Class којим добијамо објекте класе Method који садрже информације о називу, типу изузетака које метода може да баци, модификаторима, параметрима методе, повратној вредности као и методу за њен експлицитан позив. У табели 3. приказане су методе које се најчешће користе у раду са Method класом.

Табела Реч5. Метода класе Class за испитивање метода.

Метода	Опис методе
Class getDeclaringClass()	Враћа објекат класе Class који је мета-објекат класе у којој је извршена декларације методе коју представљају дати Method објекат.
Class[] getExceptionClass()	Враћа низ објеката класе Class који представљају класе изузетака које су наведене у потпису методе.
int getModifiers()	Враћа модификатор методе која је представљена датим Method објектом, трансформисаним у int.
String getName()	Враћа назив методе која је представљена датим Method објектом.
Class[] getParameterTypes()	Враћа низ објеката класе Class који представљају .class објекте параметара методе коју представљају дата Method класа
Class getReturnType()	Враћа објекат класе Class који представља тип повратне вредности методе
Object invoke(Object obj, Object[] args)	Покреће извршавање методе која је представљена датим Method објектом, над објектом који је пренет као први параметар ове методе са листом стварних параметара у оквиру низа објеката класе Object које представљају други параметар ове методе.

8.3.1 Динамичко позивање метода класа

Видели смо да се за динамичко позивање метода користи метода invoke класе Method.

```
Object invoke(Object obj, Object[] args) throws IllegalAccessException,
IllegalArgumentException;
```

Метода invoke(Object obj, Object[] args) прима два параметра, први који представља објекат над којим ће бити позвана метода, а други параметар представља низ објеката који представљају стварне аргументе методе. Уколико приступ до дате методе није дозвољен (што проверава Java Language Access Control) десиће се IllegalAccessException изузетак, а ако листа стварних аргумента не одговара потпису методе десиће се IllegalArgumentException изузетак.

Пример :

```
...
String s="Danas";
Method method = s.class.getMethod("toString",new Class[] {});
```

```
System.out.println("s ima vrednost:"+method.invoke(s, new Object[] {}));  
...  
// rezultat: s ima vrednost Danas
```

8.3.1.1 Коришћење простих типова приликом динамичог позива метода класа

Поставља се питање како проследити стварни параметар, који је простог типа, приликом позива методе. Уколико је потребно проследити параметар простог типа приликом динамичког позива методе прослеђује се обмотани објекат одговарајуће класе. На пример уколико методи треба проследити параметар типа `int` морате извршити обмотавање вредности у објекат класе `Integer`. Иста ствар важи и за повратну вредност методе. Уколико метода враћа вредност простог типа, `invoke` метода ће вратити вредност обмотану у објекат одговарајуће класе.

Пример Реф6. Дефинисати класу `AutomatNovca` која има атрибуте `imeKlijenta` типа `String`, `stanje` типа `double`, методу `prikazi` која приказује тренутно стање, и методе `ulaganje` и `podizanje` које као параметар примају износ за који се повећава односно скида новац са стања. Креирати објекат класе `AutomatNovca` и дати пример динамичког позива метода.

```
package ref6;  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;  
import javax.security.auth.AuthPermission;  
/**  
 * @author Vojislav Stanojevic  
 * SILAB - Labartorija za Softversko Inzenjerstvo  
 * FON - Beograd  
 * http://silab.fon.bg.ac.yu  
 */  
public class Ref6 {  
  
    /** Creates a new instance of ref6 */  
    public Ref6() {  
    }  
    public static void main(String[] args) {  
        AutomatNovca au = new AutomatNovca("Zika Zikic",200.00);  
        Class metaAN = au.getClass();  
        try {  
            Method metUlaganje = metaAN.getDeclaredMethod("ulaganje",new Class[]{double.class});  
            metUlaganje.invoke(au,new Object[]{new Double(150)});  
  
            Method metPodizanje = metaAN.getDeclaredMethod("podizanje",new Class[]{double.class});  
            metPodizanje.invoke(au,new Object[]{new Double(100)});  
  
            Method metPrikazi = metaAN.getDeclaredMethod("prikazi",new Class[]{});  
            // moglo je i ovako  
            //Method metPrikazi = metaAN.getDeclaredMethod("prikazi",null);  
            metPrikazi.invoke(au,new Object[]{});  
  
            } catch (IllegalArgumentException ex) {  
                ex.printStackTrace();  
            } catch (IllegalAccessException ex) {  
                ex.printStackTrace();  
            } catch (InvocationTargetException ex) {  
                ex.printStackTrace();  
            }catch (SecurityException ex) {  
                ex.printStackTrace();  
            } catch (NoSuchMethodException ex) {  
                ex.printStackTrace();  
            }  
        }  
    }  
  
    class AutomatNovca{  
  
        String imeKlijenta;  
        double stanje;  
  
        AutomatNovca(String imeKlijenta,double stanje){  
            this.imeKlijenta = imeKlijenta;  
            this.stanje=stanje;
```

```

}
void prikazi(){
    System.out.println("Stanje na racunu klijenta "+imeKlijenta+" je "+stanje);
}

void ulaganje(double iznos){
    this.stanje+=iznos;
}

void podizanje(double iznos){
    if (stanje-iznos<0){System.out.println("Nije dozvoljen minus na racunu.Transakcija
odbijena!");}
    else{stanje-=iznos;}
}
}

//Rezultat:
// Stanje na racunu klijenta Zika Zikic je 250.0

```

8.4. Динамичко учитавање и креирање објеката коришћењем рефлексије

Често је тек у времену извршавања програма познато која ће од неког броја класа бити инстанцирана. Заправо неке од класа бивају учитане тек у току извршавања програма. Метода `forName` класе `Class` омогућава функционалност учитавања нове класе (уколико она наравно већ није учитана) у времену извршавања програма. Коришћењем класе `Class` могуће је извршити и динамичко креирање инстанце класе коју представља дати објекат.

8.4.1 Основе `forName` методе

Поред тога што се `Class` објекат одговарајуће класе може добити наредбом `Ime_Klase.class` постоји још један начина да се добије одговарајући `Class` објекат. Класа `Class` поседује static методу `forName(String nazivKlase)` која враћа објекат класе `Class`, где `nazivKlase` представља пуни назив класе (укључујући и целу хијерархију пакета у ком се налази жељена класа). Разлика између ова два начина креирања `Class` објекта је што коришћењем методе `forName` назив класе може бити познат тек у времену извршења.

Редослед корака приликом извршавања `forName` методе је следећи:

- Проверава се да ли је тражена класа већ учитана. Уколико јесте одмах се враћа одговарајући објекат класе `Class`
- Ако класа није учитана тражи се одговарајућа `.class` датотека на `classpath` путањи, `class loader` учитава одговарајућу класу (односно њен бајт код) и креира се одговарајући објекат класе `Class` а метода враћа референцу на тај објекат.
- У случају да не постоји класа са специфицираним именом десиће се `ClassNotFoundException`.

Пример:

```

...
String nazivKlase = "java.lang.String";
Class stringClass = Class.forName(nazivKlase)
...

```

Напомена: `Class` објекти за просте типове не могу се добити коришћењем методе `forName`. Сходно томе наредба `Class.forName(char.class.getName())` баца `ClassNotFoundException` изузетак.

8.4.2 Креирања инстанце класе коришћењем `newInstance` методе, класе `Class`

Када је потребно креирати објекат класе коришћењем рефлексије, најчешће се користи `newInstance` метода класе `Class`. Овом методом могуће је креирати појављивање само ако класа има не параметарски конструктор.

Пример:

```
...
String nazivKlase = "java.lang.String";
Class stringClass = Class.forName(nazivKlase);
String instanca = (String) stringClass.newInstance();
...
```

Уколико се у не параметарском конструктору класе деси нека грешка, биће генерисан и бачен `InstantiationException` изузетак.

8.4.3 Креирања инстанце класе коришћењем `Constructor` класе

Поред позива `newInstance` методе класе `Class`, могуће је креирати објекат одређене класе коришћењем `java.lang.reflect.Constructor` класе. Класа `Class` поседује методе за рад са конструкторима. Ове методе су аналогне методама за рад са методама и атрибутима. У табели 3.2.1. дат је преглед и кратак опис ових метода.

Табела Реф6 Методе класе `Class` за рад са конструкторима.

Метода	Опис методе
<code>Constructor getConstructor (Class[] parameterTypes)</code>	Враћа објекат класе <code>Constructor</code> који представља јавни конструктор, класе која је представљена датим <code>Class</code> објектом, са формалним параметрима који одговарају другом параметру који прослеђен овој методи
<code>Constructor [] getConstructors()</code>	Враћа низ објеката класе <code>Constructor</code> који представља све јавне конструкторе класе које представља дати <code>Class</code> објекат
<code>Constructor getDeclaredConstructor (Class[] parameterTypes)</code>	Враћа објекат класе <code>Constructor</code> који представља конструктор, класе која је представљена датим <code>Class</code> објектом, са формалним параметрима који одговарају другом параметру који прослеђен овој методи
<code>Constructor [] getDeclaredConstructors()</code>	Враћа низ објеката класе <code>Constructor</code> који представља све конструкторе класе које представља дати <code>Class</code> објекат

Уколико не постоји конструктор чији формални параметри одговарају по броју, редоследу и типу, листи параметара прослеђених методи, `getConstructor` или `getDeclaredConstructor` метода бацују `NoSuchMethodException` изузетак. Приликом рада са конструктором класом могуће је да обе ове поменуте методе баце `SecurityException`, уколико је забрањен преглед конструктора од стране менџера сигурности.

`Constructor` класа је врло слична `Method` класи са разликом што уместо `invoke` методе постоји `newInstance` метода. У табели 3.2.1. дат је преглед и кратак опис ове методе класе `Constructor`.

Табела Реф7. Методе класе `Constructor` за рад конструкторима класе.

Метода	Опис методе
....	Исте методе као код класе <code>Method</code>
<code>Object newInstance (Object [] initargs)</code>	Позива извршење конструктора, са наведеним параметрима, и враћа ново креирano појављивање класе.

Пример Реф7. Дати пример креирања објекта класе `Osoba` коришћењем `Class` и `Constructor` класе.

```
package ref7;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;

/***
 * @author Vojislav Stanojevic
 */
```

```

* SILAB - Labartorija za Softversko Inzenjerstvo
* FON - Beograd
* http://silab.fon.bg.ac.yu
*/
public class Ref7 {

    /** Creates a new instance of Ref7 */
    public Ref7() {
    }
    public static void main(String[] args) {
        Osoba o;
        try {

            o = Osoba.class.newInstance();
            System.out.println("Podaci o Osobi kreiranom sa newInstance metodom klase Class");
            o.prikazi();
            Constructor con = Osoba.class.getDeclaredConstructor(new
                Class[]{String.class, String.class, int.class});
            o = (Osoba) con.newInstance(new Object[] { "0707984811756", "Petar Petrovic",
                new Integer(22) });
            System.out.println("Podaci o Osobi kreiranom sa newInstance
                metodom klase Constructor");
            o.prikazi();

        } catch (IllegalArgumentException ex) {
            ex.printStackTrace();
        } catch (SecurityException ex) {
            ex.printStackTrace();
        } catch (InvocationTargetException ex) {
            ex.printStackTrace();
        } catch (IllegalAccessException ex) {
            ex.printStackTrace();
        } catch (InstantiationException ex) {
            ex.printStackTrace();
        } catch (NoSuchMethodException ex) {
            ex.printStackTrace();
        }
    }
}

class Osoba {
    String jmbg;
    String imePrezime;
    int godineStarosti;

    /** Creates a new instance of Osoba */
    public Osoba() {
        jmbg = "111111111111";
        imePrezime="nema";
        godineStarosti=0;
    }

    public Osoba(String jmbg, String imePrezime, int godineStarosti) {
        this.jmbg = jmbg;
        this.imePrezime=imePrezime;
        this.godineStarosti=godineStarosti;
    }

    public void prikazi(){
        System.out.println("Osoba:"+imePrezime+", sa jmbg-om "+jmbg+", stara je "+
godineStarosti);
    }
}
//Rezultat:
//Podaci o Osobi kreiranom sa newInstance metodom klase Class
//Osoba:nema, sa jmbg-om 111111111111, stara je 0
//Podaci o Osobi kreiranom sa newInstance metodom klase Constructor
//Osoba:Petar Petrovic, sa jmbg-om 0707984811756, stara je 22

```

8.5 Пролазак кроз хијархију наслеђивања

Интересантно је да не постоји метода која би могла да нам врати све чланице класе, и оне које су дефинисане у оквиру класе и оне које су наслеђене од надкласа.

Наиме метода `getMethod` враћа све методе класе (и декларисане и оне које су наслеђене) али не враћа методе до којих је забрањен приступ. Са друге стране `getDeclaredMethods` враћа све методе које су декларисане у оквиру класе, али не и оне које је класа наследила, без обзира на право приступа до методе. И свега наведеног може се извести закључак да морамо да прођемо кроз целу хијерархију класа како би добили чланице подкласе⁴⁵.

8.5.1 Испитивање хијерархије наслеђивања

Добијање референце на `.class` објекат надкласе неке класе је само једна од операција које обезбеђује Јавин API за рефлексију, у раду са хијерархијом наслеђивања. У табели 4 је приказан скуп метода за рад са хијерархијом класа.

Tabela Реф8. Методе класе `Class` за испитивање хијерархије наслеђивања.

Метода	Опис методе
<code>Class[] getInterfaces()</code>	Враћа низ објеката класе <code>Class</code> који представља директне интерфејсе ⁴⁶
<code>Class getSuperclass()</code>	Враћа објеката класе <code>Class</code> који представља директну надкласу ⁴⁷ датог <code>Class</code> објекта или <code>null</code> уколико је дати објект представља <code>Object</code> класу.
<code>Boolean isAssignableFrom(Class cls)</code>	Враћа <code>true</code> ако и само ако класа или интерфејс који је представљен овим <code>Class</code> објектом или иста/и, или надинтерфејс класе коју представља пренети <code>Class</code> параметар
<code>Boolean isInstance(Object obj)</code>	Враћа <code>true</code> ако и само ако је класа чији је објекат пренет као параметар методе компатибилан са класом представљеном датим <code>Class</code> објектом

Метода `getInterfaces` враћа низ `Class` објеката који представљају интерфејсе. Када се ова метода позове на `Class` објекту који репрезентује класу, низ ће садржати све интерфејсе које имплементира дата класа (интерфејсе наведене у `implements` исказу дефиниције класе). Са друге стране, када се говори о `Class` објекту који репрезентује интерфејс, ова метода враћа списак интерфејса наведених у `extends` клаузули дефиниције класе.

Пример Реф8. Дефинисати класу `X` са атрибутима `x1,x2` који су целобројног типа и класу `Y` која наслеђује класу `X` и има додатне атрибуе `y1, y2` који су целобројног типа. Креирати објекат класе `Y` и приказати вредности свих атрибута коришћењем механизма рефлексије.

```
package ref8;

import java.lang.reflect.Field;

/**
 * @author Vojislav Stanojevic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

public class Ref8 {
    /** Creates a new instance of Ref8 */
    public Ref8() {
    }
}
```

⁴⁵ Аналогно је и код класе `Field`

⁴⁶ Представља интерфејсе које имплементира само дата класа не и интерфејсе које су имплементирале класе које су хијерархији изнад ње

⁴⁷ Представља директну надкласу, односно класу која је наведена у `extends` клаузули

```

public static void main(String[] args) {
    Y objY = new Y();
    objY.x1=2;
    objY.x2=2;
    objY.y1=10;

    Field [] atributi;
    Class pomocna = Y.class;
    System.out.println("Objekat objY ima sledece vrednosti:");
    while (pomocna!=null){
        atributi = pomocna.getDeclaredFields();
        for (int i = 0; i < atributi.length; i++) {
            if (!atributi[i].isAccessible()) {atributi[i].setAccessible(true);}
            try {

                System.out.println(atributi[i].getName()+"="+atributi[i].getInt(objY));

            } catch (IllegalArgumentException ex) {
                ex.printStackTrace();
            } catch (IllegalAccessException ex) {
                ex.printStackTrace();
            }
        }
        pomocna=pomocna.getSuperclass();
    }
}

class X{
    int x1;
    int x2;
}

class Y extends X{
    int y1;
    int y2;
}

//Rezultat:
//Objekat objY ima sledece vrednosti:
//y1=10
//y2=0
//x1=2
//x2=2

```

8.6. Рад са мета-подацима табела у бази

Поред механизма рефлексије који нам омогућава рад са мета подацима класе постоје и класе помоћу којих можемо добити информације о структури бази и њеним табелама. Приликом креирања упита над неком табелом у бази из `ResultSet`-а могуће је добити мета податке о табели над којом је извршен упит⁴⁸. Пакет `java.sql` има интерфејсе `DataBaseMetaData` и `resultSetMetaData`, у којима су дефинисане методе помоћу којих добијамо мета информације о бази и свакој од табела. Коришћењем конкретних класа које реализују ове интерфејс⁴⁹, могу се добити информације о називу шеме, типовима података, функцијама које постоје, информација о примарним кључевима, колонама табеле као и типовима над којима су дефинисане...

Пример ДБМета1. Креирати базу података `student` у MySQL СУБП. У креираној бази направити табелу `Student` која има поља `brind`, `muna String`, `ime muna String` и `prezime muna String`. Приказати информације о табели `Student`.

```
package DBMeta;
```

⁴⁸ Информације садрже само податке оних поља која су наведена у `Select` исказу.

⁴⁹ Свака од `jar` датотека садржи класу која имплементира наведене интерфејсе, па сходно томе имамо ситуацију да неке реализације ових интерфејса не имплементирају све методе, што доводи до тога да је корисник ускраћен за те информације.

```

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
/**
 * @author Vojislav Stanojevic
 * SILAB - Labartorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */
public class DBMetal {

    /** Creates a new instance of RsMetal */
    public DBMetal() {
    }
    public static void main(String[] args) {
        try {
            String dbUrl = "jdbc:mysql://127.0.0.1:3306/student";
            String user = "root";
            String pass = "";
            Class.forName("com.mysql.jdbc.Driver");
            Connection naredba = DriverManager.getConnection(dbUrl, user, pass);
            System.out.println("Uspostavljena je konekcija izmedju driver manager-a i baze");

            String upit = "SELECT * FROM STUDENT";
            Statement st = naredba.createStatement();
            ResultSet rs = st.executeQuery(upit);
            ResultSetMetaData rsMeta = rs.getMetaData();

            //System.out.println("Tabela Student pripada "+rsMeta.getSchemaName()+" semi");
            System.out.println("Table ima "+rsMeta.getColumnCount()+" kolone");

            System.out.println("*****");
            DatabaseMetaData dbMeta = naredba.getMetaData();
            ResultSet rsDB = dbMeta.getPrimaryKeys(null,null,"Student");

            System.out.println("Primarni kljuc sacinjavaju kolone:");
            while (rsDB.next()){
                System.out.println("Naziv kolone:"+rsDB.getString("COLUMN_NAME"));
            }
            System.out.println("*****");
            System.out.println("Kolone tabele Student:");
            System.out.println("*****");
            for (int i = 1; i <= rsMeta.getColumnCount(); i++) {
                System.out.println("Kolona "+rsMeta.getColumnName(i)+" , je "
                        +rsMeta.getColumnTypeName(i));
                if (rsMeta.isAutoIncrement(i)) {System.out.println(", kolona ima autoinkrement");}
            }
        } catch (ClassNotFoundException cnfe) {
            System.out.println("Nije ucitan upravljacki program: " + cnfe);
        } catch (SQLException sqle) {
            System.out.println("Greska: " + sqle);
        }
    }
}

//Rezultat:
//Table ima 3 kolone
//*****
//Primarni kljuc sacinjavaju kolone:
//Naziv kolone:brind
//*****
//kolone tabele Student:
//Kolona brind , je VARCHAR
//Kolona ime , je VARCHAR
//Kolona prezime , je VARCHAR

```

Задаци

Реф31. Дефинисати класе Sportista (надкласа) и Kosarkas (подкласа). Приказати информације о свим чланицама класе Kosarkas.

Реф32. Дефинисати класу `Kosarkas` и показати како се преносе параметри приликом динамичког позива методе.

Реф33. Дефинисати класе `Sportista` (надкласа) и `Kosarkas` (подкласа), и на примеру показати како из класе Кошаркаш приступа до не јавних чланица класе `Sportista`.

Реф34. Дефинисати класе `Sportista` (надкласа) и `Kosarkas` (подкласа), и на примеру показати динамичко инстанцирање објеката ових класа коришћењем `Constructor` класе.

ДБМета31. Креирати базу података `student` у MySQL СУБП. У креираној бази направити табелу `Predmet` која има поља `sifraPredmeta` које је ауто инкремент, назив предмета типа `String`, и `semestar` типа `int`. Приказати основне информације о табели `Predmet`.

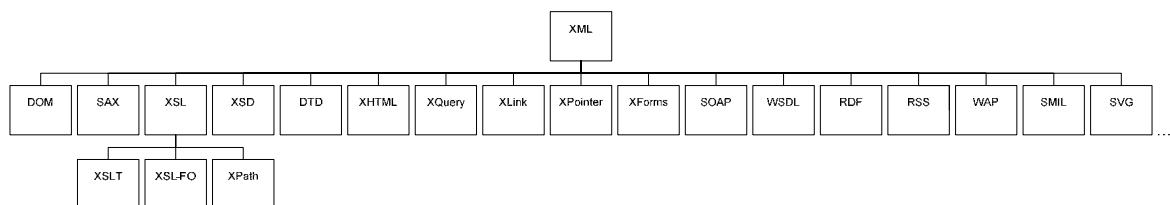
Питања:

1. Да ли прости типови имају `.class` мета-објекат?
2. Да ли постоји и која је то ако постоји, метода којом можемо добити списак свих метода једне класе (и дефинисане и наслеђене)?
3. Како добијамо информације о надкласи?
4. Како се преноси параметар простог типа приликом позива `invoke` методе?
5. Како се позива метода која нема параметре?
6. Која метода се користи за стопирање менаџера контроле приступа?
7. Чему служи `forName` метода?
8. Како се креира објекат класе која има само параметарски конструктор?
9. Који интерфејси дефинишу методе за приступ мета подацима базе?
10. У којим пакетима су реализовани интерфејси `DatabaseMetaData` и `ResultSetMetaData`?

9. ЈАВА И XML

XML (EXtensible Markup Language) је прошириви језик за означавање структуре докумената. *XML* је пројектован да буде лако читљив и представља препоруку *W3C (World Wide Web Consortium)* конзорцијума од фебруара 1998. године [W3S].

XML је, као и *HTML*, језик за означавање али им се намена битно разликује. Наиме, првенствена намена *HTML*-а је описивање начина приказивања података док се *XML*-ом описује структура података. Пошто је платформски независан, *XML* омогућава размену структурираних података између различитих информационих система и у последње време налази изузетно велику примену. На слици XML1 приказане су *XML* технологије [W3S]. Као што се може видети, *XML* је постао основа других језика: *XHTML*-а (новија верзија *HTML*-а), *WML*-а (језик за описивање начина приказивања података у мобилним уређајима), *WSDL*-а (језик за опис *web* сервиса)...



Слика XML1: XML технологије

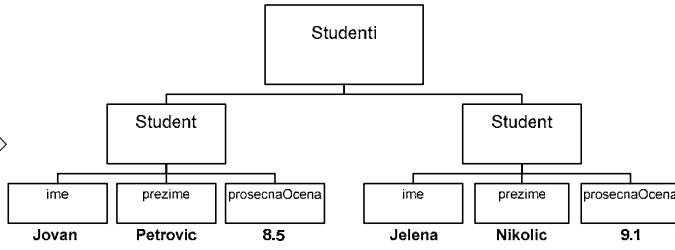
Основна структура *XML* документа је хијерархијска и такав документ се састоји од атрибута, елемената и података који су типа знаковног низа. *XML* је проширив језик, што значи да не постоје предефинисани тагови већ се морају дефинисати тагови који ће се користити у *XML* документу.

Пример XML1: Креирати *XML* документ у коме ће се чувати подаци о студентима. О сваком студенту треба чувати следеће податке: име, презиме и просечну оцену.

Могућа реализација таквог *XML* документа је следећа:

```

<?xml version="1.0" encoding="UTF-8"?>
<studenti xmlns="http://www.silab.fon.bg.ac.yu"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:SchemaLocation="silab.xsd">
    <student>
        <ime>Jovan</ime>
        <prezime>Petrovic</prezime>
        <prosečnaOcena>8.5</prosečnaOcena>
    </student>
    <student>
        <ime>Jelena</ime>
        <prezime>Nikolic</prezime>
        <prosečnaOcena>9.1</prosečnaOcena>
    </student>
</studenti>
  
```



Слика XML2: Структура XML документа

Прва линија представља *XML* декларацију. У њој се наводи верзија *XML*-а, као и кодна страна која се ће се користити. Следећа линија описује корени (почетни - *root*) елемент унутар кога се налазе сви остали елементи (у том смислу се корени елемент може посматрати као контејнер осталих елемената). Унутар овог елемента налази се елемент *student* који има три елемента унутар њега: *ime*, *prezime* и *prosečnaOcena*. Као што се може видети, *XML* документ се састоји из *XML* тагова. При томе, сваки елемент мора имати почетни таг (на пример, таг *<ime>*) и крајњи таг (на пример, *</ime>*). При томе, потребно је водити рачуна о хијерархији елемената (нпр. унутар кореног елемента

studenti налазе све сви остали елементи, тако да се овај таг затвара на крају документа). На основу тога добија се хијерархија која је приказана на слици XML2. *XML* документ мора да буде **добро оформљен**. Добро оформлен *XML* документ задовољава следеће услове:

1. Постоји декларација *XML* документа,
2. Постоји само један корени елемент унутар кога се налазе остали елементи,
3. Сви елементи и атрибути у документу морају бити синтаксно исправни, тј. сви елементи (осим празног елемента⁵⁰) морају имати почетни и крајњи таг, при чему вредност атрибута мора бити наведена унутар знакова навода.

Уколико се дефинише опис (односно тип) *XML* документа и уколико *XML* документ поштује дефинисани опис каже се да је *XML* документ **валидан**. Опис *XML* документа се може дефинисати на два начина:

1. Коришћењем *DTD-а (Document Type Definition)*
2. Коришћењем *XML* шеме (*XML Schema Definition – XSD*), која је заснована на *XML-у*. Овај начин је значајно семантички богатији и представља препоруку *W3C* конзорцијума од маја 2001. године [W3S].

9.1 Обрада XML докумената

Програмски језик Јава подржава две технологије за рад са *XML* докуменатима:

1. *JAXP (Java API for XML Processing)*. Ова технологија подржава два *API-ја* за обраду *XML* докумената [IvHo]:
 - *SAX (Simple API for XML)*
 - *DOM (Document Object Model)*
2. *JAXB (Java Architecture for XML Binding)*. Ова технологија постала је саставни део платформе Java SE 6.0 [JoZu]

9.1.1 SAX

За обраду *XML* докумената *SAX* користи приступ заснован на догађајима. Парсер пролази кроз *XML* документ, проналази делове документа (на пример, почетак документа, крај документа, почетак елемента, крај елемента) и позива методу која одговара насталом догађају (на пример, `startDocument()`, `endDocument()`, `startElement()`, `endElement()`). На слици XML3 дат је приказ једноставаног *XML* документа и догађаја који настају приликом његове обраде.

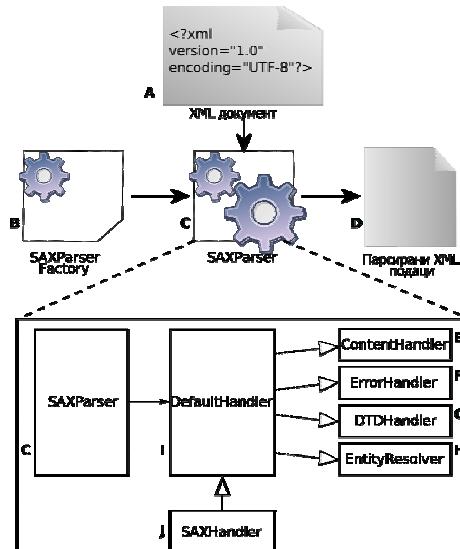
XML документ	Догађаји у програму
<pre><?xml version="1.0" encoding="UTF-8"?> <studenti> <student> <ime> Jovan Petrovic </ime> </student> </studenti></pre>	<pre>-> Почетак XML документа -> Почетак елемента: studenti -> Почетак елемента: student -> Почетак елемента: ime -> Знакови: Jovan Petrovic -> Крај елемента: ime -> Крај елемента: student -> Крај елемента: studenti -> Крај XML документа</pre>

Слика XML3: Догађаји при обради *XML* документа (*SAX* парсер)

⁵⁰ Елемент је празан уколико нема садржај. Празан елемент представља се почетним тагом иза кога непосредно следи крајњи таг (на пример `<ime></ime>`) или тагом за означавање празног елемента (на пример таг `<ime/>`).

Клase које омогућавају рад са *SAX* парсером налазе се у пакету `org.xml.sax`. Поред тога, помоћне класе налазе се у пакету `org.xml.sax.helpers`, док се додатне класе налазе у пакету `org.xml.sax.ext` (на пример, класе за прикупљање података о *DTD* дефиницији *XML* документа). Такође, у пакету `javax.xml.parsers` налазе се класе за креирање појављивања (инстанце) парсера.

Принцип рада *SAX* парсера представљен је на слици XML4. Потребно је обрадити *XML* документ (**A**) како би се добили парсирани *XML* подаци (**D**). Приликом рада најпре је потребно креирати инстанцу класе `SAXParserFactory` (**B**) која је, у ствари, задужена за креирање инстанце *SAX* парсера (**C**). Приликом позива методе `parse()` позивају се методе које одговарају насталим догађајима (на пример, `startDocument()`, `endDocument()`, `startElement()`, `endElement()`). Ове методе дефинисане су у интерфејсима `ContentHandler` (**E**), `ErrorHandler` (**F**), `DTDHandler` (**G**) и `EntityResolver` (**H**). Класа која имплементира методе поменутих интерфејса је `DefaultHandler` (**I**). Намеће се закључак да је за обраду одређеног догађаја потребно прекрити одговарајућу методу поменуте класе (**J**).



Слика XML4: Принцип рада *SAX* парсера

Пример XML2: За *XML* документ који има структуру као на слици XML3 приказати садржај коришћењем *SAX* парсера.

```

package yu.ac.bg.fon.silab.xml.sax;

/*
 * @author Milos Milic
 * SILAB - Laboratoriya za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

//SAXParserFactory -> SAXParser -> XML datoteka, SAXHandler -> Parsirani XML podaci

import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class SaxPrimer {
    public static void main(String[] args) {
        File xmlFile = new File("studenti.xml"); //A
        process(xmlFile);
    }
}
  
```

```
private static void process(File file) {
    // klasa za kreiranje instance SAX parsera
    SAXParserFactory spf = SAXParserFactory.newInstance(); //B
    SAXParser parser = null;
    try {
        // kreiranje instance SAX parsera
        parser = spf.newSAXParser(); //C
        System.out.println("Parser: " + parser.getClass().getName());
    } catch (SAXException e) {
        e.printStackTrace();
    } catch (ParserConfigurationException e) {
        e.printStackTrace();
    }
    System.out.println("Pocetak parsiranja XML datoteke: " + file);
    // handler klasa za obradu dogadjaja
    SaxHandler handler = new SaxHandler(); //J
    try {
        // parsiranje XML dokumenta
        parser.parse(file, handler);
    } catch (IOException e) {
        e.printStackTrace();
    } catch (SAXException e) {
        e.printStackTrace();
    }
}
}

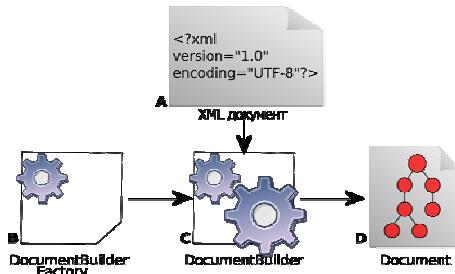
class SaxHandler extends DefaultHandler { //I, J
    public void startDocument() {
        System.out.println("Pocetak XML dokumenta");
    }
    public void endDocument() {
        System.out.println("Kraj XML dokumenta");
    }
    public void startElement(String uri, String localName, String qname,
                            Attributes attr) {
        System.out.println("Pocetak elementa: " + qname);
        int attrCount = attr.getLength();

        // ispisuju se podaci o atributima elementa (ukoliko postoje)
        if (attrCount > 0) {
            System.out.println("Atributi:");
            for (int i = 0; i < attrCount; i++) {
                System.out.println("Naziv: " + attr.getQName(i));
                System.out.println("Tip: " + attr.getType(i));
                System.out.println("Vrednost: " + attr.getValue(i));
            }
        }
    }

    public void endElement(String uri, String localName, String qname) {
        System.out.println("Kraj elementa: " + qname);
    }
    public void characters(char[] ch, int start, int length) {
        System.out.println("Znakovi: " + new String(ch, start, length));
    }
    public void ignorableWhitespace(char[] ch, int start, int length) {
        System.out.println("Prazan prostor: " + new String(ch, start, length));
    }
}
//Rezultat:
//Parser: com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl
//Pocetak parsiranja XML datoteke: studenti.xml
//Pocetak XML dokumenta //D
//Pocetak elementa: studenti
//Pocetak elementa: student
//Pocetak elementa: ime
//Znakovi: Jovan Petrovic
//Kraj elementa: ime
//Kraj elementa: student
//Kraj elementa: studenti
//Kraj XML dokumenta
```

9.1.2 DOM

За разлику од *SAX*-а, приликом обраде *XML* документа *DOM* парсер у меморији креира хијерархијску структуру која је у апликацији доступна као појављивање класе *Document*⁵¹ (слика XML5). На овај начин је могуће приступити свим елементима *XML* документа. Поред тога, *DOM* има још једну значајну предност у односу на *SAX*: **DOM дозвољава измену и креирање нових докумената.**



Слика XML5: Принцип рада *DOM* парсера

Класе које омогућавају рад са *DOM* парсером налазе се у пакету `org.w3c.dom` и одговарајућим потпакетима. Као што је већ напоменуто, у пакету `javax.xml.parsers` налазе се класе за креирање појављивања парсера.

Пример XML3: Приказати начин обраде *XML* документа (структурата документа дата је на Слици XML3) коришћењем *DOM* парсера. Проверити да ли је *XML* документ добро оформљен. Добро оформлен *XML* документ задовољава следеће услове:

1. Постоји декларација *XML* документа,
2. Постоји само један корени елемент унутар кога се налазе остали елементи,
3. Сви елементи и атрибути у документу морају бити синтаксно исправни, тј. сви елементи (осим празног елемента) морају имати почетни и крајњи таг, при чему вредност атрибута мора бити наведена унутар знакова навода.

```

package yu.ac.bg.fon.silab.xml.dom;

/*
 * @author Milos Milic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

//DocumentBuilderFactory -> DocumentBuilder -> XML datoteka -> Document

import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;

public class DomPrimer {
    public static void main(String[] args) {
        File xmlFile = new File("studenti.xml"); //A
        DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance(); //B
        DocumentBuilder builder = null;
        try {
            builder = builderFactory.newDocumentBuilder(); //C
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        }
        System.out.println("Builder Factory = " + builderFactory.getClass().getName());
    }
}

```

⁵¹ *Document* је, у ствари, реализован као интерфејс.

```

System.out.println("Builder = " + builder.getClass().getName());
try {
    Document xmlDoc = null;
    xmlDoc = builder.parse(xmlFile); //D
    System.out.println("XML dokument je dobro оформљен.");
} catch (SAXException e) {
    System.out.println("XML dokument nije dobro оформљен.");
    e.printStackTrace();
} catch (IOException e) {
    System.out.println("IO greška.");
    e.printStackTrace();
}
}
//Rezultat:
//Builder Factory = com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderFactoryImpl
//Builder = com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderImpl
//XML dokument je dobro оформљен.

```

За пролаз кроз *XML* документ користи се интерфејс *Node* који се налази у пакету *org.w3c.dom*. Он, у ствари, учаурује (енкапсулира) све компоненте *XML* документа. Шта више, и интерфејс *Document* је изведен из овог интерфејса.

Пример XML4: Приказати садржај *XML* документа (структурата документа је дата на слици XML3) коришћењем *DOM* парсера.

```

package yu.ac.bg.fon.silab.xml.dom;

/*
 * @author Milos Milic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

//DocumentBuilderFactory -> DocumentBuilder -> XML datoteka -> Document

import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;

public class DomPrimerList {

    public static void main(String[] args) {
        DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = null;
        try {
            builder = builderFactory.newDocumentBuilder();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
            System.exit(0);
        }

        File xmlFile = new File("studenti.xml");

        try {
            Document xmlDoc = null;
            xmlDoc = builder.parse(xmlFile);
            // prikazi sve elemente dokumenta
            listNodes(xmlDoc.getDocumentElement(), "");
        } catch (SAXException e) {
            e.printStackTrace();
            System.exit(1);
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(0);
        }
    }

    static void listNodes(Node node, String razmak) {

```

```

// Prikaz podataka o trenutnom cvoru
String nodeName = node.getNodeName();
String value = node getNodeValue();
int nodeType = node.getNodeType();
System.out.println(razmak + "Cvor: " + nodeName);
System.out.println(razmak + "Tip: " + nodeType);
System.out.println(razmak + "Vred:" + value);

// rekurzivan poziv metode kako bi se prikazao sadrzaj XML dokumenta
NodeList list = node.getChildNodes();
if (list.getLength() > 0) {
    for (int i = 0; i < list.getLength(); i++) {
        listNodes(list.item(i), razmak + " ");
    }
}
}

//Rezultat:
//Cvor: studenti
//Tip: 1
//Vred:null
// Cvor: student
// Tip: 1
// Vred:null
//   Cvor: ime
//   Tip: 1
//   Vred:null
//     Cvor: #text
//     Tip: 3
//     Vred:Jovan Petrovic

```

Поред тога, за приказ садржаја *XML* документа могуће је користити и интерфејс *Element* (такође је изведен из интерфејса *Node*).

Пример XML5: Утврдити колико се елемената налази у *XML* документу (структурата документа је дата на Слици *XML3*). Приказати елементе на стандарданом излазу.

```

package yu.ac.bg.fon.silab.xml.dom;

/*
 * @author Milos Milic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

//DocumentBuilderFactory -> DocumentBuilder -> XML datoteka -> Document

import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class DomPrimerList {

    public static void main(String[] args) {
        DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = null;
        try {
            builder = builderFactory.newDocumentBuilder();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
            System.exit(0);
        }
        File xmlFile = new File("studenti.xml");
        try {
            Document xmlDoc = null;
            xmlDoc = builder.parse(xmlFile);
            // prikazi sve elemente dokumenta
            listNodes(xmlDoc);
        } catch (SAXException e) {
            e.printStackTrace();
            System.exit(0);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```

        System.exit(0);
    }

    static void listNodes(Document doc) {
        // dobijanje liste cvorova na osnovu naziva taga (*=koren)
        NodeList list = doc.getElementsByTagName("*");
        System.out.println("Broj elemenata u XML dokumentu: " + list.getLength());
        System.out.println("XML elementi: ");
        for (int i = 0; i < list.getLength(); i++) {
            // Uzimanje elementa
            Element element = (Element) list.item(i);
            System.out.println("Cvor: " + element.getNodeName());
            System.out.println("Tip: " + element.getNodeType());
        }
    }
}

//Rezultat:
//Broj elemenata u XML dokumentu: 3
//XML elementi:
//Cvor: studenti
//Tip: 1
//Cvor: student
//Tip: 1
//Cvor: ime
//Tip: 1

```

Како што је већ напоменуто, коришћењем *DOM* парсера могуће је и креирање *XML* документа. У том смислу можемо рећи да се креирање *XML* документа састоји из два корака:

1. Креирање **Document** објекта који представља структуру *XML* документа,
2. Креирање хијерархије документа коришћењем метода инстанце класе креиране у кораку 1.

Пример XML6: Креирати структуру *XML* документа који је представљен на слици *XML3* и приказати га на стандардном излазу.

```

package yu.ac.bg.fon.silab.xml.dom;

/*
 * @author Milos Milic
 * SILAB - Laboratorijska za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

//DocumentBuilderFactory -> DocumentBuilder -> XML datoteka -> Document

import java.io.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.*;

public class DomPrimerGenerateXML {
    public static void main(String[] args) throws Exception {
        createXML();
    }
    static void createXML() throws Exception {
        DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
        Document document = documentBuilder.newDocument();

        // kreiranje korenog elementa i dodavanje u dokument
        Element korenElement = document.createElement("studenti");
        document.appendChild(korenElement);

        // kreiranje elementa student
        Element emStudent = document.createElement("student");

        // kreiranje elementa ime i postavljanje vrednosti u cvor
        Element emIme = document.createElement("ime");

```

```

emIme.appendChild(document.createTextNode("Jovan Petrovic"));

// dodavanje elemenata (kreiranje strukture)
emStudent.appendChild(emIme);
korenElement.appendChild(emStudent);

// prikaz dokumenta na standardnom izlazu
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(document);
StreamResult result = new StreamResult(System.out);
transformer.transform(source, result);
}

//Rezultat:
//<?xml version="1.0" encoding="UTF-8" standalone="no"?>
//<studenti>
//  <student>
//    <ime>Jovan Petrovic</ime>
//  </student>
//</studenti>

```

Поред тога, у верзији *JAXP 1.4* додат је још један *API* за обраду *XML* докумената: *StAX* (*Streaming API for XML*). Његов принцип рада је сличан *SAX*-у, али, за разлику од њега, дозвољава и креирање *XML* докумената. Постоје две дела *StAX API*-ја [JoZu]:

1. *Cursor API*: омогућава пролаз кроз *XML* документ од почетка до краја;
2. *Iterator API*: омогућава обраду догађаја по редоследу настајања у извornom документу.

Одговарајуће класе налазе се у пакету `javax.xml.stream` и његовим потпакетима.

Пример XML7: Приказати пролаз кроз *XML* документ (структурата је дата на Слици *XML3*) коришћењем *StAX API*-ја. Приказати пролаз преко *Cursor API*-ја а затим и преко *Iterator API*-ја.

```

package yu.ac.bg.fon.silab.xml.stax;

/*
 * @author Milos Milic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

//Cursor:  XMLInputFactory -> XMLStreamReader -> XML datoteka -> XML podaci
//Iterator: XMLInputFactory -> XMLEventReader -> XML datoteka -> XMLEvent

import java.io.FileReader;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.events.XMLEvent;

public class PrimerStax {
    public static void main(String[] args) throws Exception {
        staxCursor();
        staxIterator();
    }
    static void staxCursor() throws Exception {
        System.out.println("Cursor:");
        XMLInputFactory xmlif = XMLInputFactory.newInstance();
        XMLStreamReader xmlsr = xmlif.createXMLStreamReader(new FileReader(
                "studenti.xml"));
        int eventType;
        while (xmlsr.hasNext()) {
            eventType = xmlsr.next();
            switch (eventType) {
                case XMLEvent.START_ELEMENT:
                    System.out.println(xmlsr.getName());
                    break;
                case XMLEvent.CHARACTERS:

```

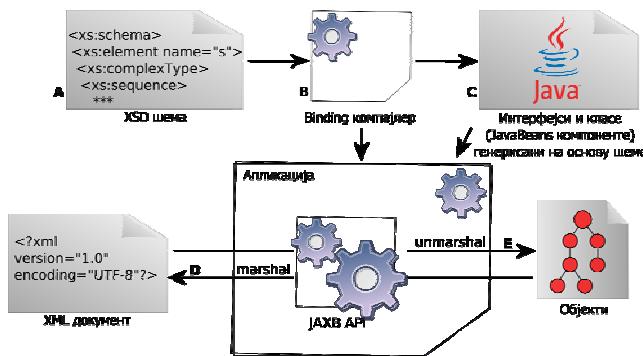
```

        System.out.println(xmlsr.getText());
        break;
    default:
        break;
    }
}
static void staxIterator() throws Exception {
    System.out.println("Iterator:");
    XMLInputFactory xmlif = XMLInputFactory.newInstance();
    XMLEventReader xmler = xmlif.createXMLEventReader(new FileReader(
        "studenti.xml"));
    XMLEvent event;
    while (xmler.hasNext()) {
        event = xmler.nextEvent();
        if (event.isStartElement()) {
            System.out.println(event.asStartElement().getName());
        } else if (event.isCharacters()) {
            System.out.println(event.asCharacters().getData());
        }
    }
}
//Rezultat:
//Cursor:
//studenti
//student
//ime
//Jovan Petrovic
//Iterator:
//studenti
//student
//ime
//Jovan Petrovic

```

9.1.3 JAXB

Ова технологија омогућава пресликање *JavaBeans* компоненти у *XML* документе и супротно. Компанија *Sun Microsystems* дефинише Јава зрна (*JavaBeans*) као „софтверске компоненте, са могућношћу поновног коришћења, којима се може визуелно манипулисати у одговарајућем алату“ [GrHa]. *JavaBeans* компоненте су, у ствари, Јава класе са одговарајућим *set* и *get* методама. Принцип рада *JAXB* технологије приказан је на Слици XML6. На основу *XSD* шеме (A), позивом *Binding* компјултера (B), генеришу се интерфејси и класе (*JavaBeans* компоненте - C) које су у складу са дефинисаном шемом. Након тога, могуће је извршити пресликање *JavaBeans* компоненти у *XML* документе (D) и обрнуто (E).



Слика XML6: Принцип рада *JAXB* технологије

Класе које омогућавају рад са *JAXB* технологијом налазе се у пакету *javax.xml.bind* и његовим потпакетима.

Пример XML8: Креирати XML документ на основу датог појављивања класе студент. Класа студент има следеће атрибуте: име, презиме и просек. Сачувати XML документ у датотеци izlaz.xml.

```
package yu.ac.bg.fon.silab.xml.jaxb;

/*
 * @author Milos Milic
 * SILAB - Laboratorijska za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

// JAXBContext -> Marshaller -> Student -> XML datoteka

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.annotation.XmlRootElement;
import java.io.*;

// potrebno je da se klasa Student oznaci (antotira) sa @XmlElement
@XmlRootElement
class Student { //C
    private String ime;
    private String prezime;
    private double prosek;

    // default konstruktor mora da postoji
    public Student() {
        ime = new String("nema");
        prezime = new String("nema");
        prosek = 5;
    }
    public Student(String ime, String prezime, double prosek) {
        this.ime = ime;
        this.prezime = prezime;
        this.prosek = prosek;
    }
    public String getIme() {
        return ime;
    }
    public void setIme(String ime) {
        this.ime = ime;
    }
    public String getPrezime() {
        return prezime;
    }
    public void setPrezime(String prezime) {
        this.prezime = prezime;
    }
    public double getProsek() {
        return prosek;
    }
    public void setProsek(double prosek) {
        this.prosek = prosek;
    }
}
public class PrimerJAXBMarshaller {
    public static void main(String[] args) {
        try {
            JAXBContext context = JAXBContext.newInstance(Student.class);
            Marshaller m = context.createMarshaller();
            m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
            Student p = new Student("Jovan", "Petrovic", 9);
            // preslikavanje objekta p u XML dokument - cuvanje u datoteci izlaz.xml
            m.marshal(p, new FileWriter("izlaz.xml")); //D
        } catch (JAXBException jex) {
            System.out.println("JAXB greska: " + jex);
            jex.printStackTrace();
        } catch (IOException e) {
            System.out.println("IO greska: " + e);
            e.printStackTrace();
        }
    }
}

//Rezultat:
```

```
//<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
//<student>
//    <ime>Jovan</ime>
//    <prezime>Petrovic</prezime>
//    <prosek>9.0</prosek>
//</student>
```

Поступак пресликања Јава класе у *XML* документ назива се кодирање (*marshaling*). Са друге стране, поступак трансформације *XML* документа у Јава класу назива декодирање (*unmarshaling*) [JoZu].

Пример XML9: На основу генерисаног *XML* документа у претходном примеру креирати појављивање класе *студент*. Приказати податке о студенту на стандардном излазу.

```
package yu.ac.bg.fon.silab.xml.jaxb;

/*
 * @author Milos Milic
 * SILAB - Laboratoriya za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

// JAXBContext -> Unmarshaller -> XML datoteka -> Student

import java.io.*;
import javax.xml.bind.*;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
class Student { //C

    private String ime;
    private String prezime;
    private double prosek;

    public Student() {
        ime = new String("nema");
        prezime = new String("nema");
        prosek = 5;
    }
    public Student(String ime, String prezime, double prosek) {
        this.ime = ime;
        this.prezime = prezime;
        this.prosek = prosek;
    }
    public String getIme() {
        return ime;
    }
    public void setIme(String ime) {
        this.ime = ime;
    }
    public String getPrezime() {
        return prezime;
    }
    public void setPrezime(String prezime) {
        this.prezime = prezime;
    }
    public double getProsek() {
        return prosek;
    }
    public void setProsek(double prosek) {
        this.prosek = prosek;
    }
}

public class PrimerJAXBUnmarshaller {
```

```

public static void main(String[] args) throws IOException {
    try {
        JAXBContext context = JAXBContext.newInstance(Student.class);

        Unmarshaller u = context.createUnmarshaller();
        // preslikavanje iz XML datoteke u objekat klase Student
        Student s = (Student) u.unmarshal(new File("izlaz.xml")); //E
        System.out.println("**** Podaci o studentu ****");
        System.out.println("Ime : " + s.getIme());
        System.out.println("Prezime : " + s.getPrezime());
        System.out.println("Prosek : " + s.getProsek());
    } catch (JAXBException jex) {
        System.out.println("JAXB greska: " + jex);
        jex.printStackTrace();
    }
}
//Rezultat:
//**** Podaci o studentu ****
//Ime : Jovan
//Prezime : Petrovic
//Prosek : 9.0

```

Раније је поменуто да је *XML* документ валидан уколико поштује дефинисани опис. Препорука је да се за опис *XML* документа користи *XSD* шема. На слици XML7 приказана је *XSD* шема и добро оформљен, валидан *XML* документ који задовољава ту шему.

<i>XSD</i> шема <i>(silab.xsd)</i> <pre> <?xml version="1.0" encoding="UTF-8"?> <xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.silab.fon.bg.ac.yu" xmlns="http://www.silab.fon.bg.ac.yu" elementFormDefault="qualified"> <xss:element name="studenti"> <xss:complexType> <xss:sequence> <xss:element name="student" maxOccurs="unbounded"> <xss:complexType> <xss:sequence> <xss:element name="ime" type="xs:string"/> <xss:element name="prezime" type="xs:string"/> <xss:element name="prosečnaOcena" type="xs:decimal"/> </xss:sequence> <xss:attribute name="godina" type="xs:int"/> </xss:complexType> </xss:element> </xss:sequence> </xss:complexType> </xss:element> </xss:schema> </pre>	<i>XML</i> документ <i>(studenti.xml)</i> <pre> <?xml version="1.0" encoding="UTF-8"?> <studenti xmlns="http://www.silab.fon.bg.ac.yu" xmlns: xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="silab.xsd"> <student godina="4"> <ime>Jovan</ime> <prezime>Petrovic</prezime> <prosečnaOcena>8.5</prosečnaOcena> </student> <student godina="4"> <ime>Jelena</ime> <prezime>Nikolic</prezime> <prosečnaOcena>9.1</prosečnaOcena> </student> </studenti> </pre> 
---	--

Слика XML7: *XSD* шема и валидан *XML* документ

JAXB омогућава генерирање *JavaBeans* компоненти које задовољавају *XSD* шему. У ту сврху користи се алат *xjc* који се покреће из командне линије.

Пример XML10: Показати како се на основу *XSD* шеме генеришу *JavaBeans* компоненте. Кao *XSD* шему искористити датотеку *silab.xsd* чија је структура приказана на слици XML7. Креирати појављивање компоненте, уписати је у *XML* документ и након тога је прочитати из *XML* документа.

```

> xjc silab.xsd //A, B
parsing a schema...
compiling a schema...
yu\ac\bg\fon\silab\ObjectFactory.java //C
yu\ac\bg\fon\silab\Studenti.java //C
yu\ac\bg\fon\silab\package-info.java //C

```

Као резултат позива команде *xjc* генерисане су три класе. Једна од њих је *JavaBeans* компонента (класа *Studenti.java*). Класа *ObjectFactory.java* служи за креирање *JavaBeans*

компоненти док је класа *package-info.java* помоћна класа. У наставку је дат приказ класе *Studenti.java*.

```
package yu.ac.bg.fon.silab;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.*;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = { "student" })
@XmlRootElement(name = "studenti")
public class Studenti {

    @XmlElement(required = true)
    protected List<Studenti.Student> student;

    public List<Studenti.Student> getStudent() {
        if (student == null) {
            student = new ArrayList<Studenti.Student>();
        }
        return this.student;
    }

    @XmlAccessorType(XmlAccessType.FIELD)
    @XmlType(name = "", propOrder = { "ime", "prezime", "prosecnaOcena" })
    public static class Student {

        @XmlElement(required = true)
        protected String ime;
        @XmlElement(required = true)
        protected String prezime;
        @XmlElement(required = true)
        protected BigDecimal prosecnaOcena;
        @XmlAttribute
        protected Integer godina;

        public String getIme() {
            return ime;
        }
        public void setIme(String value) {
            this.ime = value;
        }
        public String getPrezime() {
            return prezime;
        }
        public void setPrezime(String value) {
            this.prezime = value;
        }
        public BigDecimal getProsecnaOcena() {
            return prosecnaOcena;
        }
        public void setProsecnaOcena(BigDecimal value) {
            this.prosecnaOcena = value;
        }
        public Integer getGodina() {
            return godina;
        }
        public void setGodina(Integer value) {
            this.godina = value;
        }
    }
}
```

На крају, потребно је написати класу за креирање, односно читање *XML* документа.

```
package yu.ac.bg.fon.silab.xml.jaxb;

/*
 * @author Milos Milic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

// Upis u XML: JAXBContext -> Marshaller -> Student -> XML datoteka
// Citanje iz XML-a: JAXBContext -> Unmarshaller -> XML datoteka -> Student
```

```

import java.io.*;
import java.math.*;
import javax.xml.bind.*;
import yu.ac.bg.fon.silab.Studenti;
public class PrimerXjc {
    public static void main(String[] args) throws Exception {
        try {
            Studenti studentiWrite = new Studenti();
            Studenti.Student s1 = new Studenti.Student();
            s1.setIme("Jovan");
            s1.setPrezime("Petrovic");
            s1.setProsečnaOcena(new BigDecimal(8.5));
            studentiWrite.getStudent().add(s1);

            JAXBContext context = JAXBContext.newInstance(Studenti.class);

            // upis u XML datoteku
            Marshaller m = context.createMarshaller();
            m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
            m.marshal(studentiWrite, new FileWriter(new File("studenti_xjc.xml")));

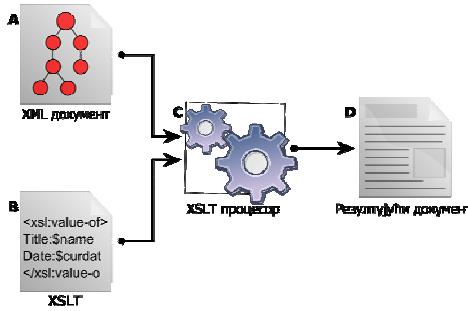
            // citanje iz XML datoteke
            Unmarshaller u = context.createUnmarshaller();
            Studenti studentiRead = (Studenti) u.unmarshal(new File(
                "studenti_xjc.xml"));
        } catch (JAXBException jex) {
            System.out.println("JAXB greska: " + jex);
            jex.printStackTrace();
        }
    }
}

```

9.2 XSLT трансформација

XSLT (EXtensible Stylesheet Language Transformations) је језик заснован на *XML*-у који омогућава трансформацију *XML* докумената у друге документе. *XSLT* се често користи за трансформацију података између различитих *XML* шема (конверзија *XML* документа у *XML* документ⁵²), као и за трансформацију података у *HTML* или *XHTML* документе. *XSLT* је препорука *W3C* конзорцијума од новембра 1999. године [W3S].

Принцип рада *XSLT* трансформације приказан је на слици XML8. На основу *XML* документа (**A**) и описа трансформације (*XSLT* документа - **B**), *XSLT* процесор (**C**) пролази кроз *XML* документ и трансформише га у одговарајући резултујући документ (**D**) [Wiki]. При томе *XSLT* процесор користи *XPath* за пролаз кроз атрибуте и елементе *XML* документа.



Слика XML8: Принцип рада *XSLT* трансформације

⁵² Ова конверзија се користи уколико је потребно да се на другачији начин представи структура постојећег *XML* документа. У том случају је *XSLT* трансформација адаптер који прилагођава структуру постојећег *XML* документа структури новог *XML* документа.

Пример XML11: Пrikазати трансформацију XML документа у XHTML документ. Структура изворног XML документа као и резултујућег XHTML документа представљена је на слици XML9.

XML документ <i>(studenti.xml)</i> <pre><?xml version="1.0" encoding="UTF-8"?> <studenti xmlns="http://www.silab.fon.bg.ac.yu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="silab.xsd"> <student godina="4"> <ime>Jovan</ime> <prezime>Petrovic</prezime> <prosecnaOcena>8.5</prosecnaOcena> </student> <student godina="4"> <ime>Jelena</ime> <prezime>Nikolic</prezime> <prosecnaOcena>9.1</prosecnaOcena> </student> </studenti></pre>	XHTML документ <i>(studenti.html)</i> <pre><?xml version="1.0" encoding="utf-8"?> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> <html xmlns="http://www.w3.org/1999/xhtml"> <head> <title>XSLT primer</title> </head> <body> <h1>Studenti</h1> <table border="1"> <tr> <td>Jovan</td><td>Petrovic</td><td>8.5</td> </tr> <tr> <td>Jelena</td><td>Nikolic</td><td>9.1</td> </tr> </table> </body> </html></pre>
--	---



Слика XML9: Трансформација XML документа у XHTML документ

Најпре је потребно дефинисати XSLT датотеку на основу које ће се вршити трансформација XML документа у XHTML документ. У наставку је дат приказ XSLT датотеке *transform.xsl*.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/1999/xhtml">
<xsl:template match="/studenti">
<xsl:output doctype-public=""-//W3C//DTD XHTML 1.0 Strict//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" indent="yes">
</xsl:output>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>XSLT primer</title>
</head>
<body>
<h1>Studenti</h1>
<table border="1">
<xsl:apply-templates select="student">
</xsl:apply-templates>
</table>
</body>
</html>
</xsl:template>
<xsl:template match="student">
<tr>
<td><xsl:value-of select="ime"/></td>
<td><xsl:value-of select="prezime"/></td>
<td><xsl:value-of select="prosecnaOcena"/></td>
</tr>
</xsl:template>
</xsl:stylesheet>
```

Након тога могуће је извршити трансформацију документа и у ту сврху се користи класа Transformer која се налази у пакету *javax.xml.transform*.

```
package yu.ac.bg.fon.silab.xml.xslt;

/*
 * @author Milos Milic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

// TransformerFactory -> Transformer -> DOMSource -> StreamResult
```

```

import java.io.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.*;
import org.w3c.dom.Document;
public class PrimerXsltXhtml {
    public static void main(String[] args) {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        Document document = null;
        try {
            File f = new File("studenti.xml"); //A

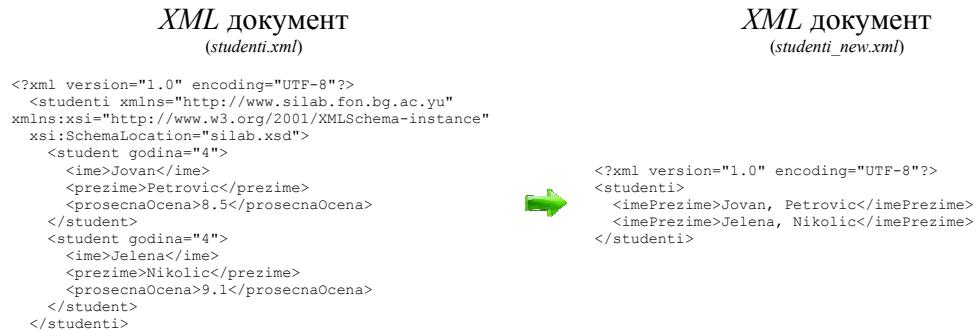
            // kreiranje strukture XML dokumenta (koriscenje DOM parsera)
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(f);

            // XSL datoteka
            File styleSheet = new File("transform.xsl"); //B
            StreamSource styleSource = new StreamSource(styleSheet);

            // kreiranje Transformer klase
            TransformerFactory tFactory = TransformerFactory.newInstance();
            Transformer transformer = tFactory.newTransformer(styleSource);
            DOMSource source = new DOMSource(document);
            StreamResult result = new StreamResult(new FileOutputStream("studenti.html")); //D
            // transformacija
            transformer.transform(source, result); //C
        } catch (Exception e) {
            System.out.println("Greska: " + e);
            e.printStackTrace();
        }
    }
}

```

Пример XML12: Приказати трансформацију XML документа у XML документ. Структура извornog XML документа као и резултатујућег XML документа представљена је на слици XML10.



Слика XML10: Трансформација XML документа у XML документ

Најпре је потребно дефинисати *XSLT* датотеку на основу које ће се вршити трансформација XML документа у XML документ. У наставку је дат приказ *XSLT* датотеке *transform_xml.xsl*.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/">
    <studenti> <xsl:apply-templates/> </studenti>
  </xsl:template>

  <xsl:template match="//student">
    <imePrezime>
      <xsl:value-of select="ime" />, <xsl:value-of select="prezime" />
    </imePrezime>
  </xsl:template>
</xsl:stylesheet>

```

Након тога могуће је извршити трансформацију документа.

```
package yu.ac.bg.fon.silab.xml.xslt;

/*
 * @author Milos Milic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * http://silab.fon.bg.ac.yu
 */

// TransformerFactory -> Transformer -> DOMSource -> StreamResult

import java.io.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.*;
import org.w3c.dom.Document;

public class PrimerXsltXml {

    public static void main(String[] args) {

        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        Document document = null;
        try {
            File f = new File("studenti.xml");

            // kreiranje strukture XML dokumenta (koriscenje DOM parsera)
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(f);

            // XSL datoteka
            File styleSheet = new File("transform_xml.xsl");
            StreamSource styleSource = new StreamSource(styleSheet);

            // kreiranje Transformer klase
            TransformerFactory tFactory = TransformerFactory.newInstance();
            Transformer transformer = tFactory.newTransformer(styleSource);

            DOMSource source = new DOMSource(document);
            StreamResult result = new StreamResult(new FileOutputStream("studenti_new.xml"));
            // transformacija
            transformer.transform(source, result);

        } catch (Exception e) {
            System.out.println("Greska: " + e);
            e.printStackTrace();
        }
    }
}
```

XML је платформски и технолошки независтан језик који је брзо постао стандард у размени података између хетерогених апликација, технологија и окружења. *XML* представља стандард и препоруку *W3C* конзорцијума. Његова једноставност, самоописивост (*self-descriptiveness*) и стално усавршавање омогућавају најразличитије примене. Шта више, *XML* је постао основа других језика и саставни део многих спецификација технологија. Веома је значајна примена *XML*-а у реализацији *Web* сервиса.

9.3 Питања

1. Шта је *XML*?

2. У чему је разлика између *XML*-а и *HTML*-а?
3. Објаснити структуру *XML* документа. Дати пример.
4. Шта је добро оформљен *XML* документ?
5. Шта је валидан *XML* документ? Како се може дефинисати опис *XML* документа?
6. Које технологије се могу користити за обраду *XML* документа? Објаснити разлике.
7. Шта је *XSLT* трансформација? Објаснити начин рада.

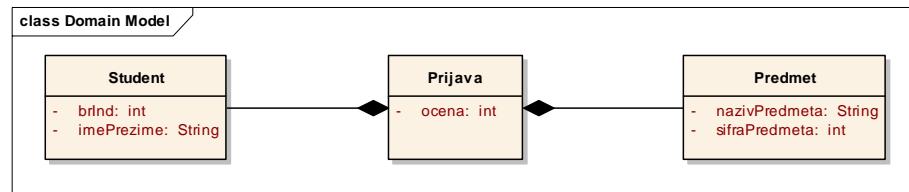
9.4 Задаци за вежбу

XMLZ1.Дат је *XML* документ чија је структура представљена на Слици XML10. На основу просечне оцене сваког студента израчунати просечну оцену студената четврте године. Резултат приказати на стандардном излазу. За обраду *XML* документа користити *DOM* парсер.

XMLZ2.Дат је *XML* документ чија је структура представљена на Слици XML10. На стандардном излазу приказати податке о студентима чија је просечна оцена већа од 8.5. Поред тога, у бази података сачувати податке о студентима чија је просечна оцена већа од 9. За обраду *XML* документа користити *DOM* парсер.

XMLZ3.Осмислiti генерички пример креирања **добро оформленог *XML*** документа и додавања произвoльнog броja елемената. Документ приказати на стандардном излазу и сачувати у *XML* датотеци. За обраду *XML* документа користити *DOM* парсер.

XMLZ4.Дат је следећи *UML* дијаграм класа:



Креирати пријаве о полаганим испитима у текућем року и сачувати их у *XML* датотеци. Након тога пријаве прочитати из *XML* датотеке и приказати их на стандардном излазу. Користити *JAXB* технологију.

XMLZ5.За претходни пример креирати *XSD* шему и генерисати потребне класе. Након тога пријаве прочитати из *XML* датотеке и приказати их на стандардном излазу. Користити *JAXB* технологију.

XMLZ6.Дата је *XML* датотека у којој се налазе подаци о полаганим испитима студената у текућем испитном року. Пronađi и приказати студенте који су добили оцену 10 из предмета Пројектовање софтвера. Користити *JAXB* технологију.

XMLZ7.Дата је *XML* датотека у којој се налазе подаци о полаганим испитима студената у текућем испитном року. Креирати извештај о положеним испитима у *HTML* формату који садржи име и презиме студената који

ни су положили испит Пројектовање софтвера. За креирање извештаја користити *XSLT* трансформацију.

XMLZ8. Познато је да се *RSS* (*Really Simple Syndication*) технологија заснива на *XML*-у. Са сајта Радио-Телевизије Србије учитати *RSS* који прати вести из Науке и културе (тренутно је то <http://www.rts.co.yu/statd/rssnauka.xml>) и креирати табеларни извештај у *HTML* формату тако да се у првој колони табеле налази наслов (таг *title*), у другој линк до ресурса (таг *link*) а у трећој колони опис ресурса (таг *description*). За креирање извештаја користити *XSLT* трансформацију.

10. WEB СЕРВИСИ

Web сервиси су технологија која решава проблем позива удаљених метода. Web сервиси представљају низ протокола и стандарда који се користе за размену информација између резличитих апликација коришћењем WEB-а. [JJU]

У последњој деценији појавило се много технологија које решавају проблем позива удаљених метода (*Remote Procedure Call - RPC*). Једна од најзначајнијих међу њима јесте *Remote Method Invocation (RMI)*, која је представљала основу за наредне RPC технологије.

Remote Method Invocation (RMI)

RMI уводи концепт интерфејса на тај начин што сваки серверски објекат излаже интерфејс преко кога потенцијални клијенти могу позвати удаљену методу. Такође, постоји једноставан сервис за регистрацију серверског објекта и проналажење серверског објекта од стране клијената. RMI скрива од програмера конверзију позива обичне методе у позив удаљене методе, што значајно умањује напор потребан за реализацију удаљеног позива. RMI је платформски независна⁵³ самим тим што је потпуно развијена Јава технологијом, али је могу користити само програми написани Јава програмским језиком. Зато се за RMI каже да је *language-specific* технологија.

Common Object Request Broker Architecture (CORBA)

Велики искорак, нарочито са аспекта језичке зависности, направљен је појавом технологије *Common Object Request Broker Architecture (CORBA)*. За ову технологију направљена је подршка у Јава технологији, али и у многим другим програмским језицима, и тиме је добијена основа за интероперабилност између апликација развијених у различитим технологијама. CORBA се нарочито показала корисном у повезивању са апликацијама које су застареле чиме се стекла могућност њихове надградње модерним технологијама. Интерфејс који описује сервис написан је у новом, неутралном језику који се зове *Interface Definition Language (IDL)*. Комуникација се остварује коришћењем посебног протокола - *Internet Inter-Orb Protocol (IOP)*.

Web Services

Web сервиси су RPC технологија која је данас најактуелнија. Основна разлика између web сервиса и осталих RPC технологија је у томе што је web сервис технологија базирана на XML језику, и што за комуникацију користи стандардне *Internet* протоколе као што су HTTP, SMTP и FTP. До данас је развијено много технологија које су имале задатак да омогуће постојање web сервиса, а данас су се три технологије издвојиле као стандардне, и заједно омогућавају креирање, функционисање и проналажење web сервиса. То су *Simple Object Access Protocol (SOAP)*, *Web Service Definition Language (WSDL)* и *Universal Description, Discovery, and Integration (UDDI)*.

Simple Object Access Protocol (SOAP)

SOAP представља протокол за размену структуираних информација у дистрибуираном окружењу. SOAP користи XML за структуирање порука које могу бити размењиване кроз низ стандардних Интернет технологија (укључујући SMTP, HTTP и FTP протоколе). Тиме што SOAP користи стандардизовани транспортни механизам, омогућена је интероперабилност разнородних клијената и сервера. SOAP је протокол најсличнији IOP протоколу који користи CORBA, са основном разликом што се XML

⁵³ Платформски независне технологије – технологије које не зависе од специфичности оперативних система на којима се извршавају.

користи као текстуални документ, за разлику од ИОР протокола који користи IDL у бинарном облику. SOAP такође подржава и RPC механизам, у чијој је основи такође XML.

Web Service Definition Language (WSDL)

WSDL је XML технологија која служи за опис интерфејса *web* сервиса. Њиме се стандардизује описивање удаљене методе, улазних и излазних параметара, и на основу њега клијенти знају како треба да изврше удаљени позив.

Universal Description, Discovery and Integration (UDDI)

UDDI дефинише начин за објављивање и проналажење *web* сервиса. Представља једну врсту јединственог регистратора *web* сервиса, који потенцијалним клијентима пружа могућност да се на једном месту упознају са свим регистрованим *web* сервисима, претражујући их по различитим критеријумима. UDDI пружа информације о типу функционалности коју пружа *web* сервис, и о локацији и начину како му се може приступити. Важно је напоменути да *web* сервис не мора нужно бити објављен на UDDI регистратору, али тада клијенти на неки други начин морају бити обавештени о постојању, и локацији *web* сервиса.

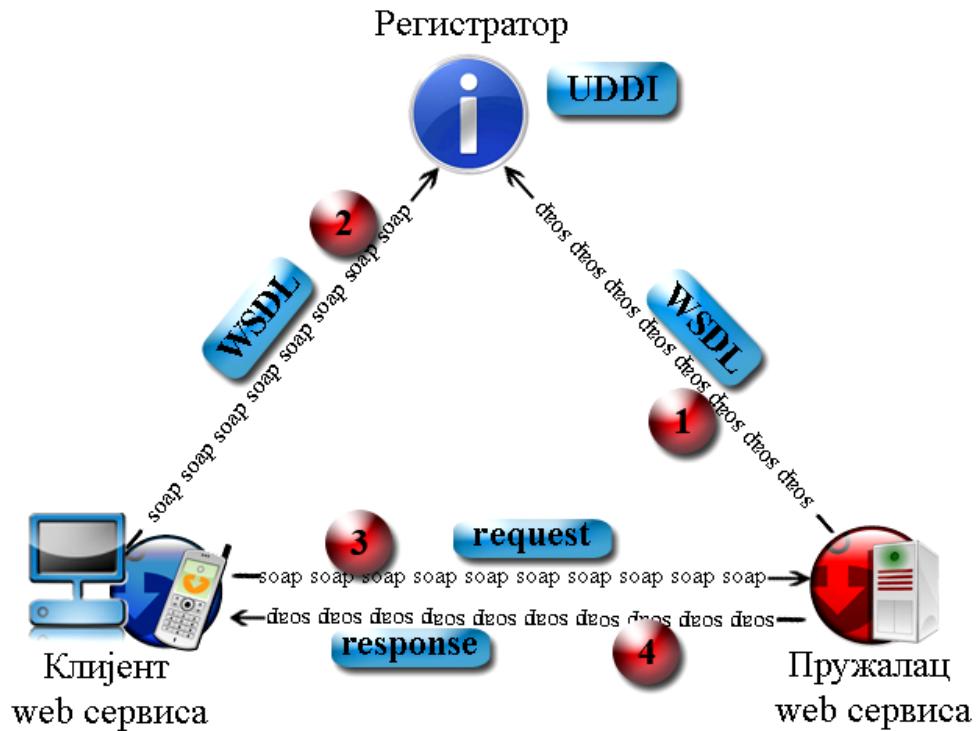
Животни циклус web сервиса

При функционисању *web* сервиса издвајају се три основне улоге:

- Клијент *web* сервиса (*Service Requester*), апликација (*Web*, *Desktop*, или *Web сервис*) која позива извршење методе *web* сервиса.
- Пружалац *web* сервиса (*Service Provider*), сервер на коме је подигнут *web* сервис.
- Регистратор *web* сервиса (*Service Broker*), сервер преко кога клијенти добијају могућност претраге постојећих *web* сервиса, и информације о коришћењу сервиса.

Најчешћи сценарио који се среће код креирања и функционисања *web* сервиса одвија се на следећи начин (Слика WS1):

1. Пружалац *web* сервиса подиже *web* сервис и региструје информације о сервису на неком од постојећих Регистратора *web* сервиса, најчешће на UDDI. За опис сервиса користи се WSDL, а за протокол у комуникацији SOAP.
2. Клијент врши претрагу Регистратора како би пронашао одговарајући *web* сервис, и добија информације (WSDL) о томе како и где треба да позове извршење методе *web* сервиса.
3. Клијент на основу WSDL-а упућује позив за извршење Пружаоцу сервиса (коришћењем SOAP протокола).
4. Пружалац сервиса извршава захтев, и резултат шаље Клијенту (коришћењем SOAP протокола).



Слика WS1: Упрошћен приказ web сервис архитектуре

10.1 Web сервиси у Java SE 6–JAX WC

До појаве Java SE 6, библиотеке за креирање web сервиса биле су део *Enterprise* едиције (Java EE 5), да би се сада нашле у њеном стандардном издању (Java SE 6). JAX-WS2.0 представља библиотеку која садржи све потребне елементе за креирање и приступ web сервисима (укључујући *Java Architecture for XML Binding* (JAXB) и подршку за SOAP). Основне класе за развој web сервиса подељене су у три пакета:

javax.jws	Садржи метаподатке web сервиса које је путем анотација ⁵⁴ могуће придржити класама и на тај начин развијати web сервис. Анотирање се обавља тако што се класи, методи или атрибуту придржуји <code>@tag</code> и на тај начин се тој чланици придржи одређени метаподатак, на основу којег програм који врши парсирање може да јој придржи одговарајућу функционалност. Парсер који се користи код креирања web сервиса јесте <code>wsgen</code> ⁵⁵ и испоручује се уз стандардно Java издање. Java уводи анотације почев од верзије Java SE 5, а у Java 6 су анотације додатно проширене.
javax.xml.SOAP	Садржи класе и интерфејсе који чине <i>SOAP with Attachments API for Java</i> (SAAJ). Можемо га поделити у две логичке целине: класе за примање и слање порука и класе које се баве структуром самих порука.
javax.xml.ws	Садржи класе и интерфејсе који представљају основу JAX-WS API-a.

⁵⁴ Анетирати - означити

⁵⁵ `wsgen.exe` је програм који се испоручује уз стандардно Java издање, и налази се у директоријуму `$JAVA_HOME/bin` где је `$JAVA_HOME` инсталациони директоријум JDK-а.

Важно је нагласити да је цела имплементација конвертовања обичног позива методе у позив удаљене методе скривена од програмера, тако да програмер не мора бити упознат са начином рада са SOAP технологијом, па чак ни са WSDL језиком. У Java SE 6 постоје алати који све помоћне класе и датотеке аутоматски генеришу. То су алати: *wsgen* и *wsimport*. У наставку ће на једноставном примеру бити објашњен начин њиховог коришћења. Такође, са овом Java верзијом испоручује се једноставан web сервер који је погодан за тестирање web сервиса.

10.1.1 Кораци при креирању web сервиса

На једном простом примеру показаћемо поступак креирања, и објављивања web сервиса.

WSZ1:

Кориснички захтев: Направити web сервис који ће за два унета броја клијентима вратити поруку о њиховом односу.

Тест пример:

Први број	Други број	Очекивани резултат
3	5	5>3
2	2	2=2

корак 1. Креирање класе

Важно је напоменути да класа мора бити придружена неком пакету јер у противном креирање web сервиса неће бити могуће.

```
package ws;

/*
 * @author Ilija Antovic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * HTTP://silab.fon.bg.ac.yu
 */

public class KomparatorBrojeva {

    public String uporediBrojeve(int prviBroj, int drugiBroj) {
        String rez=null;
        if(prviBroj>drugiBroj){
            rez=prviBroj+">"+drugiBroj;
        }
        if(prviBroj==drugiBroj){
            rez=prviBroj+"="+drugiBroj;
        }
        if(drugiBroj>prviBroj){
            rez=drugiBroj+">"+prviBroj;
        }
        return rez;
    }
}
```

корак 2. Додавање анотације

Постоји два основна тага којима се може анонтирати класа, а најчешће се користе @WebService и @WebMethod тагови. Њима се могу додати и остали тагови из пакета javax.jws. У коду који следи болдоване су наредбе које служе за анотацију.

```
package ws;

import javax.jws.WebMethod;
import javax.jws.WebService;

/*
 * @author Ilija Antovic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * HTTP://silab.fon.bg.ac.yu
 */

@WebService
public class KomparatorBrojeva {

    @WebMethod
    public String uporediBrojeve(int prviBroj, int drugiBroj) {
        String rez=null;

        if(prviBroj>drugiBroj){
            rez=prviBroj+">"+drugiBroj;
        }

        if(prviBroj==drugiBroj){
            rez=prviBroj+"="+drugiBroj;
        }

        if(drugiBroj>prviBroj){
            rez=drugiBroj+">"+prviBroj;
        }

        return rez;
    }
}
```

корак 3. Генерирање помоћних класа wsgen алатом

Wsgen.exe је алат који се испоручује уз JDK1.6.x⁵⁶, и налази се на локацији где је инсталiran JDK, у bin поддиректоријуму. Овај алат служи за парсирање анонтиране класе, и креирање помоћних класа неопходних за функционисање web сервиса. Пре него што се позове извршење wsgen наредбе, потребно је превести (искомпајлирати) написани код:

```
javac ws/KomparatorBrojeva.java
```

Затим треба извршити wsgen наредбу на следећи начин:

```
wsgen -cp . ws.KomparatorBrojeva
```

⁵⁶ JDK1.6.x – Где x представља актуелно издање (на пр: JDK1.6.01, JDK1.6.02...)

Резултат извршења наредбе јесте креирање новог пакета `jaxws` који садржи датотеке:

- `UporediBrojeve.java`
- `UporediBrojeve.class`
- `UporediBrojeveResponse.java`
- `UporediBrojeveResponse.class`

корак 4. Објављивање web сервиса на web сервер

Након што смо креирали web сервис, потребно га је “објавити” на неком web серверу како би могао бити доступан клијентима. JDK 1.6.x садржи уgraђени web сервер на који ћемо објавити креирани web сервис. За ову намену направићемо класу `KomparatorBrojevaPublisher` у чијој `main` методи ће бити позвана наредба која ће покренути web сервер и на њега поставити web сервис.

```
package ws;

import javax.xml.ws.Endpoint;

/*
 * @author Ilija Antovic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * HTTP://silab.fon.bg.ac.yu
 */

public class KomparatorBrojevaPublisher {

    public static void main(String[] args) {

        /* objavljivanje servisa na adresi:
         * HTTP://127.0.0.1:8080/KomparatorBrojevaWebService
         */

        Endpoint.publish(
                "HTTP://127.0.0.1:8080/KomparatorBrojevaWebService",
                new KomparatorBrojeva());
    }
}
```

Искомпајлирати и покренути ову класу:

```
javac ws/KomparatorBrojevaPublisher.java
```

```
java ws.KomparatorBrojevaPublisher
```

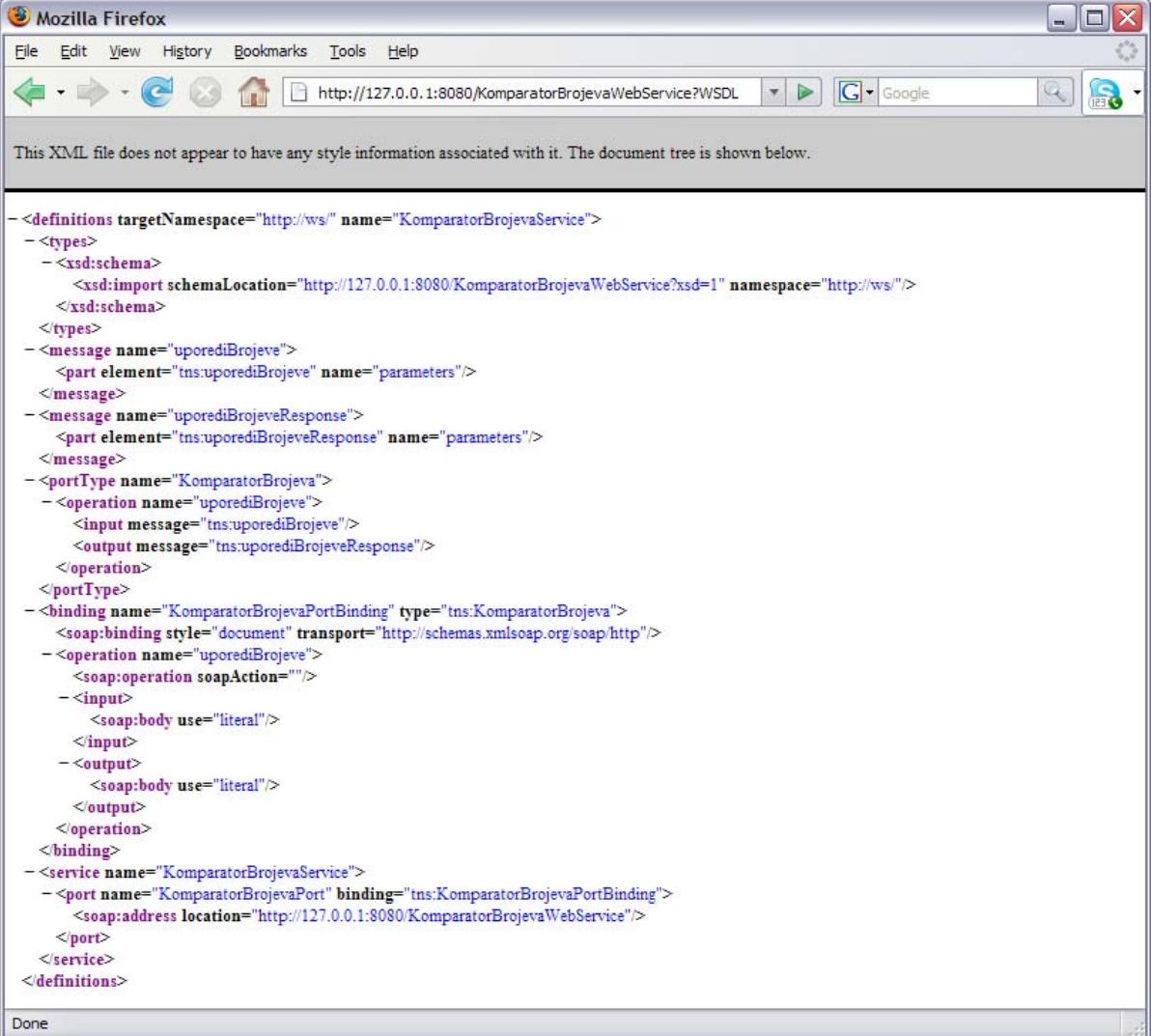
Web сервер који смо одабрали (уграђен у Java SE 6 платформу) ће бити покренут све док је Java виртуална машина активна, тако да не треба прекидати извршење класе `KomparatorBrojevaPublisher`. Пошто је овај web сервер веома скромних могућности користићемо га док не истестирамо web сервис, а касније ћемо показати како се web сервис објављује и на независним web серверима (на пр. *Apache Tomcat*)

корак 5. Тестирање web сервиса

Да би се уверили да је web сервис успешно покренут, потребно је покренути web претраживач, отићи на локацију дефинисану приликом објављивања web сервиса, са параметром WSDL:

```
http://127.0.0.1:8080/KomparatorBrojevaWebService?WSDL
```

У претраживачу би требао да се прикаже XML код који представља WSDL опис креiranог web сервиса.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<definitions targetNamespace="http://ws/" name="KomparatorBrojevaService">
  <types>
    <xsd:schema>
      <xsd:import schemaLocation="http://127.0.0.1:8080/KomparatorBrojevaWebService?xsd=1" namespace="http://ws/">
    </xsd:schema>
  </types>
  <message name="uporediBrojeve">
    <part element="tns:uporediBrojeve" name="parameters"/>
  </message>
  <message name="uporediBrojeveResponse">
    <part element="tns:uporediBrojeveResponse" name="parameters"/>
  </message>
  <portType name="KomparatorBrojeva">
    <operation name="uporediBrojeve">
      <input message="tns:uporediBrojeve"/>
      <output message="tns:uporediBrojeveResponse"/>
    </operation>
  </portType>
  <binding name="KomparatorBrojevaPortBinding" type="tns:KomparatorBrojeva">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="uporediBrojeve">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  <service name="KomparatorBrojevaService">
    <port name="KomparatorBrojevaPort" binding="tns:KomparatorBrojevaPortBinding">
      <soap:address location="http://127.0.0.1:8080/KomparatorBrojevaWebService"/>
    </port>
  </service>
</definitions>
```

Слика WS2: Садржај WSDL датотеке креiranог web сервиса



Да би се убрзalo покретањe и тестирањe web сервисa, препоручљиво јe направити извршну датотеку којом сe аутоматизујe превођењe класe, генерисањe помоћних класa и објављивањe сервисa. У корени директоријум (у коме сe налази пакет *ws*) направити датотеку *PokreniWS.bat* сa следећим садржајем:

```

PokreniWS.bat - Notepad
File Edit Format View Help
javac ws/KomparatorBrojeva.java
wsgen -cp . ws.KomparatorBrojeva
javac ws/KomparatorBrojevaPublisher.java
java ws.KomparatorBrojevaPublisher

```

10.1.2 Кораци при креирању клијента web сервисa

У наставку ћemo показати како сe врши позив извршењa методe web сервисa на примеру сервисa који смо претходно креирали и објавили. Једино што је потребно знати приликом креирањa клијентског позива јесте локацијa WSDL датотекe web сервисa кога требa позвати.

WSZ1Client:

Кориснички захтев: Написати програм који ћe позвати методу за поређењe бројева web сервисa који сe налази на локацијi:

<http://127.0.0.1:8080/KomparatorBrojevaWebService?WSDL>

Тест пример:

Први број	Други број	Очekивани резултат
3	5	Резултат позива web сервисa: 5>3
2	2	Резултат позива web сервисa: 2=2

корак 1. Генерисањe помоћних класa wsimport алатом

Wsimport.exe јe алат који сe испоручујe уз JDK1.6.x, и налази сe на локацијi где јe инсталiran JDK, у *bin* поддиректоријуму. Овај алат служи за парсирањe WSDL датотекe, и креирањe помоћних класa неопходних за позив web сервисa. Пре него што покренемо наредбу, креирајмо пакет (на пр. *wsClient*) који ћe садржати клијентски програм и помоћне датотекe. Након тога наредбу требa позвати на следећи начин:

```

wsimport -keep -p wsClient
http://127.0.0.1:8080/KomparatorBrojevaWebService?WSDL

```

Приликом позива додате су и опцијe:

-keep - трајно чувањe генерисаных датотекa

-p <package> - специфицирањe пакетa у који требa сместити генерисане датотекe

Резултат извршења наредбе јесте креирање нових .java и .class датотека унутар пакета који је специфициран (*wsClient*). Овај пакет сада садржи следеће датотеке:

- *KomparatorBrojeva.java*
- *KomparatorBrojeva.class*
- *KomparatorBrojevaService.java*
- *KomparatorBrojevaService.class*
- *ObjectFactory.java*
- *ObjectFactory.class*
- *Package-info.java*
- *Package-info.class*
- *UporediBrojeve.java*
- *UporediBrojeve.class*
- *UporediBrojeveResponse.java*
- *UporediBrojeveResponse.class*

корак 2. Писање клијентског програма за web сервис

```
package wsClient;

/*
 * @author Ilija Antovic
 * SILAB - Laboratorija za Softversko Inzenjerstvo
 * FON - Beograd
 * HTTP://silab.fon.bg.ac.yu
 */
public class KorisnikKomparatoraBrojeva {

    public static void main(String[] args) {

        /**
         * Instanciranje generisanog servisa
         */
        KomparatorBrojevaService service =
                new KomparatorBrojevaService();

        /**
         * Dobijanje instance port objekta koji cemo koristiti
         * za poziv udaljene metode
         */
        KomparatorBrojeva KomparatorProxy =
                service.getKomparatorBrojevaPort();

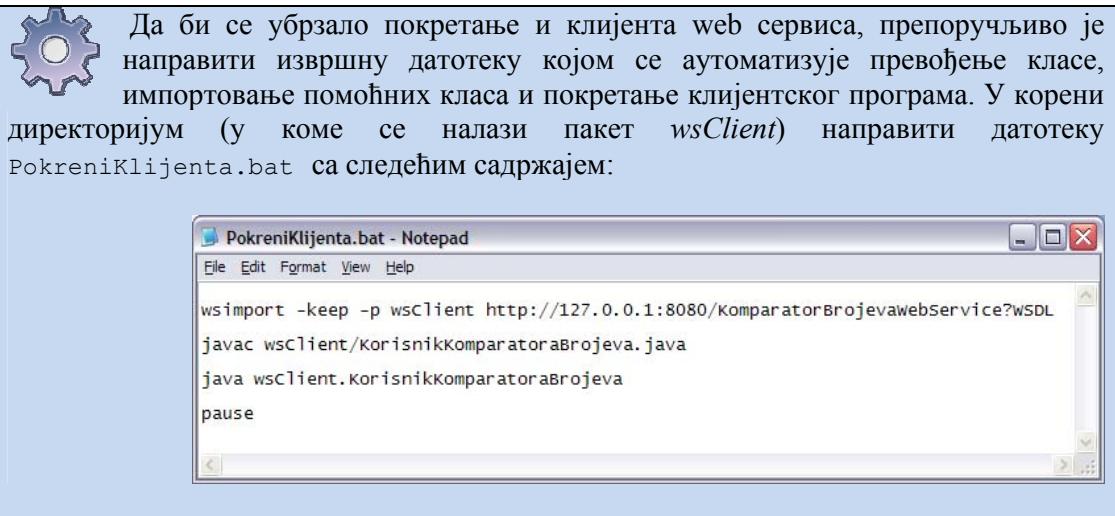
        /**
         * deklaracija i inicijalizacija argumenata
         */
        int a=3;
        int b=5;

        /**
         * Poziv udaljene metode
         */
        String result = komparatorProxy.uporediBrojeve(a, b);

        /**
         * Prikaz rezultata izvršenja udaljene metode
         */
        System.out.println("Rezultat poziva web servisa: "+result);
    }
}
```

Када се програм покрене, на екрану ће се добити следећи резултат:

Резултат позива web сервиса: 5>3



10.1.3 Објављивање web сервиса на Apache Tomcat web серверу

Web сервер који се испоручује уз Java SE 6 платформу идеалан је за тестирање web сервиса током његовог развоја, међутим овај сервер пружа слабе перформансе, па би за функционисање web сервиса ипак требало одабрати неки од проверених web сервера. Један од најкоришћенијих web сервера данас јесте *Apache Tomcat*⁵⁷, и у наставку ћемо показати на који начин се web сервис објављује на *Tomcat* серверу.

корак 1. Подешавање Tomcat сервера како би се омогућила подршка за web сервисе

- Преузети и отпаковати библиотеку класа *jaxws-2_0.jar*. Библиотека се налази на локацији: <https://jax-ws.dev.java.net/jax-ws-20-fcs/>
- Ову библиотеку треба распаковати на следећи начин:

```
java -jar jaxws-2_0.jar
```

- У директоријуму у којем су отпаковане датотеке налази се *lib* директоријум који садржи *.jar* датотеке. Све датотеке из овог директоријума треба копирати на локацију:

```
$CATALINA_HOME/lib
```

где је *\$CATALINA_HOME* – инсталациони директоријум *Apache Tomcat web сервера*.

⁵⁷ *Apache Tomcat* може се бесплатно преузети са адресе <http://tomcat.apache.org/>. Поншто је верзија JDK-а са којом радимо 1.6.x, верзија *Tomcat*-а треба да буде минимум 6.x.

корак 2. Креирање структуре директоријума за web сервис

- У директоријуму \$CATALINA_HOME/webapps направити директоријум у коме ће се наћи потребне датотеке за веб сервис. За наш пример:

```
$CATALINA_HOME/webapps/MojWebServis
```

- У креiranом директоријуму направити директоријум WEB-INF:

```
$CATALINA_HOME/webapps/MojWebServis/WEB-INF
```

- У WEB-INF директоријуму направити директоријум classes, и у њега ископирати структуру директоријума нашег web сервиса (пакет са свим класама).

корак 3. Креирање web.xml датотеке

У директоријуму WEB-INF направити датотеку web.xml која треба да има следећи садржај:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
<web-app>
    <listener>
        <listener-class>
            com.sun.xml.ws.transport.http.servlet.WSServletContextListener
        </listener-class>
    </listener>
    <servlet>
        <servlet-name>KomparatorBrojevaService</servlet-name>
        <servlet-class>
            com.sun.xml.ws.transport.http.servlet.WSServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>KomparatorBrojevaService</servlet-name>
        <url-pattern>/KomparatorBrojeva</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>60</session-timeout>
    </session-config>
</web-app>
```

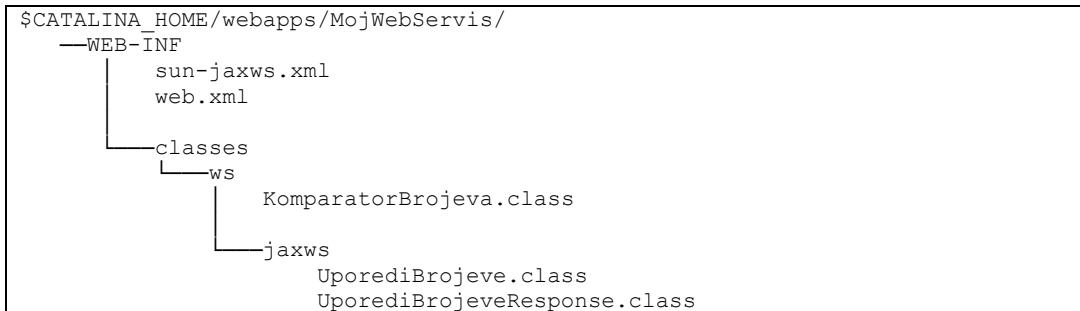
корак 4. Креирање sun-jaxws.xml датотеке

У директоријуму WEB-INF направити датотеку sun-jaxws.xml која треба да има следећи садржај:

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
    <endpoint
        name="KomparatorBrojeva"
        implementation="ws.KomparatorBrojeva"
        url-pattern="/KomparatorBrojeva"
    />
</endpoints>
```

корак 5. Проверити све датотеке

- У нашем случају након претходних корака биће следећа структуре директоријума и датотека:

**корак 6. Покренути web сервер**

У \$CATALINA_HOME/bin директоријуму налази се датотека tomcat6.exe којом се покреће web сервер. У конзоли сервера приказаће се садржај сличан следећем:

```

Feb 8, 2008 3:06:50 PM org.apache.catalina.core.AprLifecycleListener init
INFO: The Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: C:\Program Files\Apache Software Foundation\Tomcat 6.0\bin;.;C:\WINDOWS\Sun\Java\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\system32\WBem;C:\Program Files\Java\jdk1.6.0\bin;C:\Program Files\Common Files\Ahead\lib
Feb 8, 2008 3:06:50 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-8080
Feb 8, 2008 3:06:50 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 483 ms
Feb 8, 2008 3:06:50 PM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Feb 8, 2008 3:06:50 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.14
Feb 8, 2008 3:06:50 PM com.sun.xml.ws.transport.http.servlet.WSServletContextListener contextInitialized
INFO: WSSERULET12: JAX-WS context listener initializing
Feb 8, 2008 3:06:51 PM com.sun.xml.ws.transport.http.servlet.RuntimeEndpointInfoParser processWsdlLocation
INFO: wsdl cannot be found from DD or annotation. Will generate and publish a new WSDL for SEI endpoints.
Feb 8, 2008 3:06:52 PM com.sun.xml.ws.transport.http.servlet.WSServletDelegate init
INFO: WSSERULET14: JAX-WS servlet initializing
Feb 8, 2008 3:06:52 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Feb 8, 2008 3:06:52 PM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
Feb 8, 2008 3:06:52 PM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/94 config=null
Feb 8, 2008 3:06:52 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 2414 ms
  
```

Слика WS3: Конзола web сервера

корак 7. Тестирајти web сервис

Да би се уверили да је web сервис успешно покренут, потребно је покренути web претраживач, отићи на локацију дефинисану url-pattern елементом у web.xml датотеци (у нашем случају /KomparatorBrojeva) на локацији која представља контекст на web серверу (у нашем случају /MojWebServis).

```
http://127.0.0.1:8080/MojWebServis/KomparatorBrojeva
```

У претраживачу би требао да се прикаже следећи садржај:

Port Name	Status	Information
KomparatorBrojeva	ACTIVE	Address: http://localhost:8080/MojWebServis/KomparatorBrojeva WSDL: http://localhost:8080/MojWebServis/KomparatorBrojeva?wsdl Port QName: {http://ws/}KomparatorBrojevaPort Implementation class: ws.KomparatorBrojeva

Слика WS4: Изглед добијен у претраживачу након успешног покретања web сервиса

WSDL датотека налази се на локацији:

<http://localhost:8080/MojWebServis/KomparatorBrojeva?wsdl>



Поступак за креирање клијентског програма који позива методу *web* сервиса објављеног на *Tomcat web* серверу, исти је као и код *web* сервиса објављеног на уграђеном *web* серверу, те га није потребно поново објашњавати. За креирање клијентског програма једино је потребно знати локацију WSDL датотеке!

10.2 Питања

8. Које су технологије претходиле појави web сервиса?
9. У чему је разлика између web сервиса и осталих RPC технологија?
10. Које технологије чине основу web сервиса?
11. Шта је WSDL?
12. Шта је потребно клијенту да зна како би извршио позив методе web сервиса?

10.3 Задаци за вежбу

- WSV1.** Направити web сервис за конверзију новца између две валуте, објавити га на web серверу и тестирати га.
- WSV2.** Направити web сервис који приказује имена, презимена и бројеве индекса студената чији се подаци налазе у бази података. Објавити га на web сервер и тестирати.
- WSV3.** Направити клијентски програм који позива методу web сервиса за конверзију валуте.
- WSV4.** Направи клијентски програм који приказује студенте чије презиме почиње словом А. Листу студената треба добити позивом web сервиса.
- WSV5.** Направити клијентски програм који се повезује на Google web сервис за претраживање.
- WSV6.** Web сервисе из 1. и 2. задатка објавити и тестирати на Apache Tomcat web серверу.

Референце

- [J2EE14TUT] : *The J2EE 1.4 Tutorial*, Sun Microsystems, August 31, 2004
- [R2D1J2EEKOMP] : *Java Technology Concept Map*, Sun Microsystems,
<http://java.sun.com/developer/onlineTraining/new2java/javamap/intro.html>
- [WikiJavaPlatform] : http://en.wikipedia.org/wiki/Java_platform
- [JJU] JavaJazzUp magazine: <http://www.javajazzup.com/>, issue 1.
- [IJW] Introducing JAX-WS 2.0 With the Java SE 6 Platform:
http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/
- [DCTJ] David Chappell, Tyler Jewell: *Java Web Services*, O'Reilly, 2004, Sebastopol, CA, USA
- [W3S] W3Schools Online Web Tutorials: <http://www.w3schools.com/>
- [IvHo] Ivor Horton: *Ivor Horton's Beginning Java 2, JDK 5 Edition*, Wiley publishing, 2004, Indianapolis, Indiana, USA
- [JoZu] John Zukowski: *Java 6 Platform Revealed*, Apress, 2006, Berkeley, CA, USA
- [GrHa] Graham Hamilton (editor): *JavaBeans™ API specification*, Sun Microsystems, 1997, Mountain View, CA, USA
- [Wiki] Wikipedia, слободна енциклопедија: <http://en.wikipedia.org/>

