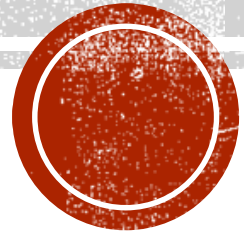
ОБЈЕКТНО ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ
ПРОГРАМСКИ ЈЕЗИК ЈАВА – 1

Опис конструкција програмског језика Јава



ГРАМАТИКА, СИНТАКСА И СЕМАНТИКА

- Граматика је наука о језику и његовим законима.
  - Она дефинише скуп правила којима се описују све валидне (исправне) конструкције, прихватљиве у језику.
- Синтакса је наука о језику у којој се изучава образовање граматички коректних конструкција језика.
  - Њу чини скуп правила за образовање правилних конструкција језика.
  - Уколико је програм написан без поштовања правила о запису конструкција програмског језика, јављају се синтаксне грешке.
- Семантика је наука о језику у којој се изучава значење конструкција језика.
  - Семантику језика чини скуп правила за утврђивање значења конструкција језика.
  - Већину семантичких грешака не може да открије преводилац, већ то мора да уради човек. Овакве грешке се, у принципу, теже откривају.

ОБЈЕКТ-ЈЕЗИК И МЕТАЈЕЗИК

- Објект-језик је језик који се изучава.
- Метајезик је језик помоћу којег се описује, односно изучава објект-језик.
- За формални (строги) опис синтаксе програмских језика најчешће се користе:
  - Бекусова нотација
  - синтаксни дијаграми
- Семантика програмских језика се, обично, формално описује помоћу:
  - Бечког дефиниционог језика или
  - Ван Вајнгаренових граматика
- У пракси се, по правилу, интуитивно описује семантика програмских језика, коришћењем природног (говорног) језика

БЕКУСОВА НОТАЦИЈА

- Синтакса програмског језика се описује помоћу коначног скупа металингвистичких формула (МЛФ)
- МЛФ се састоји из леве и десне стране раздвојене универзалним мета-симболом ::=
  - Дакле, МЛФ је облика: α ::= γ
- При чему је:
  - α металингвистичка променљива
  - γ металингвистички израз
- Металингвистичка променљива је фраза природног језика ограђена стреличастим заградама (<, >)
- Металингвистичка константа је знак или кључна реч објект-језика
- Универзални метасимбол ::= чита се „по дефиницији је“
- Универзални метасимбол | чита се „или“

БЕКУСОВА НОТАЦИЈА(2)

- Металингвистички израз може бити:
  - металингвистичка променљива
  - металингвистичка константа
  - коначан низ металингвистичких променљивих или металингвистичких константи раздвојених универзалним метасимболом | или надовезаних једни на друге

БЕКУСОВА НОТАЦИЈА(3)

- Цео декадни број дефинисан помоћу Бекусове нотације:

```
<не нула декадна цифра> ::= 1|2|3|4|5|6|7|8|9
<декадна цифра> ::= 0|<не нула декадна цифра>
<знак броја> ::= +|-
<подвлака> ::= _
<декадни цео број без знака> ::= <не нула декадна цифра>
                                |<декадни цео број без знака><декадна цифра>
                                |<декадни цео број без знака><подвлака>
<декадни цео број> ::= 0|<декадни цео број без знака>
                       |<знак броја><декадни цео број без знака>
```

БЕКУСОВА НОТАЦИЈА(4)

- Модификација изворне Бекусове нотације постиже се увођењем нових метасимбола:
  - лева и десна велика заграда ({, }) — означавају понављање обухваћене конструкције нула, једном или више пута
  - лева и десна средња заграда ([,]) — означавају опционо понављање обухваћене конструкције
  - лева и десна мала заграда ((,)) — означавају груписање конструкција

БЕКУСОВА НОТАЦИЈА(5)

- Цео декадни број дефинисан помоћу модификоване Бекусове нотације:

```
<не нула декадна цифра> ::= 1|2|3|4|5|6|7|8|9
<декадна цифра> ::= 0|<не нула декадна цифра>
<декадна цифра или подвлака> ::= <декадна цифра>|_
<декадни цео број без знака> ::= <не нула декадна цифра>{<декадна цифра или подвлака>}
<знак броја> ::= +|-
<декадни цео број> ::= [<знак броја>]0|[<знак броја>]<декадни цео број без знака>
```

- Алтернативна дефиниција целог декадног броја:

```
<не нула декадна цифра> ::= 1|2|3|4|5|6|7|8|9
<декадна цифра> ::= 0|<не нула декадна цифра>
<декадни цео број> ::= [(+|-)](0|<не нула декадна цифра>{(<декадна цифра>|_)})
```

ЕЛЕМЕНТАРНЕ КОНСТРУКЦИЈЕ ЈЕЗИКА ЈАВА

- Изворни програм језика Јава је низ Unicode знакова и он се прослеђује преводиоцу
- Преводилац анализом програма издваја наредбе, а потом елементарне конструкције
- Другим речима, елементарне конструкције су елементи језика које преводилац издваја као недељиве целине приликом превођења програма.
- У елементарне конструкције спадају: идентификатори, кључне речи, литерали, сепаратори, оператори, коментари и белине.
- Користећи Бекусову нотацију, то се записује на следећи начин:

```
<елементарна конструкција> ::= <идентификатор>|<кључна реч>|<литерал>
                               |<сепаратор>|<оператор>|<коментар>|<белина>
```

ИДЕНТИФИКАТОРИ

- Идентификатор служи за идентификовање неке конструкције у Јави
- Све конструкције у Јави, као што су: променљиве, класе, методи итд. на јединствен начин се именују преко идентификатора па се идентификатори могу поистоветити са именима
- Идентификатори имају следећу синтаксу:

```
<идентификатор> ::= (<Unicode слово>|$|_){(<Unicode слово>|$|_|<Unicode цифра>)}
```

  - Идентификатор мора почети словом, знаком за долар или подвлаком
  - У преосталом делу идентификатора, поред ових знакова, могу да се појаве и цифре
  - Јава је осетљива на величину слова, tj. у Јави постоји разлика између малих и великих слова
  - Приликом дефинисања идентификатора препоручује се избор прегледних имена

КЉУЧНЕ РЕЧИ

- Кључне речи су конструкције који имају специјалну намену у језику Јава и не могу се користити за именовање других ентитета (променљивих, класа и метода).
- Кључне речи се могу описати следећом формулом:

```
<кључна реч> ::= abstract|assert|boolean|break|byte|case|catch|char|class|const
                |continue|default|do|double|else|enum|extends|final|finally|float|
                |for|goto|if|implements|import|instanceof|int|interface|long|native|
                |new|package|private|protected|public|return|short|static|strictfp
                |super|switch|synchronized|this|throw|throws|transient|try|void
                |volatile|while
```

- Осим наведених, у Јави постоје литерали: `true`, `false` и `null` који представљају резервисане речи и не могу се користити за именовање других ентитета

ЛИТЕРАЛИ

- Литерали у језику су речи које представљају саме себе, тј. неку вредност.
- Колоквијално, литерал би се могао назвати запис константе.
- Постоје следећи типови литерала: целобројни, реални, логички, знаковни и литерали-ниске:

```
<литерал> ::= <целобројни литерал>|<реални литерал>|<логички литерал>
            |<знаковни литерал>|<литерал-ниска>
```

ЦЕЛОБРОЈНИ ЛИТЕРАЛИ

- Цели бројеви у Јави могу бити записани као декадни, октални или хексадекадни (а од верзије 7 Јаве и као бинарни)
- Почев од верзије 7, Јава допушта да целобројни литерал садржи и подвлаку
- Синтакса целбројних литерала је дата следећим МЛФ:

```
<целобројни литерал> ::= <целобројни декадни>|<целобројни октални>
                         |<целобројни хексадекадни>|<целобројни бинарни>
<целобројни декадни> ::= [<знак броја>]0
                         |[<знак броја>]<не нула декадна цифра>{(<декадна цифра>|_)}
<целобројни октални> ::= [<знак броја>]0{(<октална цифра>|_)}
<октална цифра> ::= 0|1|2|3|4|5|6|7
<целобројни хексадекадни> ::= [<знак броја>](0x|0X){(<хексадекадна цифра>|_)}
<хексадекадна цифра> ::= <декадна цифра>|a|b|c|d|e|f|A|B|C|D|E|F
<целобројни бинарни> ::= [<знак броја>](0b|0B){(0|1|_)}
```

РЕАЛНИ ЛИТЕРАЛИ

- Реални литерали су константе које се записују у облику покретне тачке
- У Јави се разликују два типа реалних литерала: `float` и `double`
- Реални литерали могу да се изразе у позиционом запису или експоненцијалном запису
- Синтакса реалних литерала је дата следећим МЛФ:

```
<реални литерал> ::= <мантиса>[<експонент><индикатор>]
<мантиса> ::= <целобројни декадни>
              |<целобројни декадни>.[{(<декадна цифра>|_)}]
              |[<знак броја>].{(<декадна цифра>|_)}
<експонент> ::= (e|E)<целобројни декадни>
<индикатор> ::= f|F|d|D
```

ЛОГИЧКИ ЛИТЕРАЛИ

- Постоје два логичка литерала представљена речима `false` и `true`:

```
<логички литерал> ::= true|false
```

- Јава је строго типизиран језик, па приликом било каквих поређења увек се као резултат добија вредност логичког типа: `true` или `false`.

ЗНАКОВНИ ЛИТЕРАЛИ

- Знаковни литерал (карактер) је било који знак осим апострофа и обрнуте косе црте, записан између апострофа.

```
<знаковни литерал> ::= <графички симбол>|' '|<ескејп-секвенца>
<графички симбол>  ::= '<знак>'
```

- Графички симбол је један занак између апосторфа (тј. једноструких наводника ')
- Празан простор (описан графичким симболом ' ') се такође уврштава у знаковне литерале.
- Ескејп-секвенце су знаковни литерали који почињу обрнутом косом цртом, после чега следи још један или више знакова

ЗНАКОВНИ ЛИТЕРАЛИ (2)

- У изворном Јава програму могу се користити графички симболи, као и ескејп-секвенце преузете из програмског језика C:

```
'\''     ► апостроф
'\"'     ► наводник
'\\'     ► обрнута коса црта
'\r'     ► знак за повратак на почетак реда (енг. carriage return)
'\n'     ► знак за прелазак у нови ред (енг. new line)
'\f'     ► знак за прелазак на нову страну (енг. form feed)
'\t'     ► знак табулатора
'\b'     ► знак за повратак за једно место уназад (енг. backspace)
```

- Пример. У Јави се користе и ескејп-секвенце којима се описују Unicode знаци:

```
'\u0041'     ► Unicode  знак, тј. слово A
'\u0161'     ► Unicode  знак, тј. слово Š
```

ЛИТЕРАЛИ-НИСКЕ

- Литерали-ниске се разликују од свих осталих јер нису литерали примитивног типа података.
- Литерал-ниска се записује као ниска знакова између наводника:

```
<литерал-ниска> ::= "{<знаковни литерал>}"
```

  - Између наводника може да се појави било који знак осим наводника и обрнуте косе црте (који могу да се појаве само у оквиру ескејп-секвенце)

- Пример. Правилан запис појединх ниски:

```
""                                          ► празна ниска (не садржи ниједан знак)
"Програмирање i matematika"                 ► непразна ниска са различитим писмима
"Ovo je navodnik \", a ovo ne \u3232"       ► ниска са ескејп-секвенцама
```

СЕПАРАТОРИ

- У Јави постоји неколико знакова који служе за раздвајање једне врсте елементарних конструкција од других
  - На пример, у сепараторе спада симбол `;` који служи за раздвање наредби у Јави
- Сепаратори су дефинисани на следећи начин:

```
<сепаратор> ::= (|)|{|}|[|]|;|:|,|.
```

- Сепаратори служе само за раздвајање и не одређују операције над подацима

ОПЕРАТОРИ И ИЗРАЗИ

- Оператори омогућавају операције над подацима.
- Подаци на које се примењују оператори називају се операнди.
- Према позицији операнада разликују се:
  - префиксни
  - инфиксни
  - постфиксни
- Према броју операнада разикују се:
  - унарни
  - бинарни
  - тернарни

ОПЕРАТОРИ И ИЗРАЗИ (2)

- Најчешће се користи подела на следеће типове оператора:

```
<оператор> ::= <аритметички оператор>|<релациони оператор>
             |<битовни оператор>|<логички оператор>|<условни оператор>
             |<инстанцни оператор>|<оператор креирања објекта>|<оператор доделе>
```

- Паралелно са изучавањем оператора обично се изучавају и одговарајући изрази
- Изрази у Јави се користе да донесу, израчунају и сместе неку вредност
- У један израз могу бити укључени:
  - операнди
  - оператори
  - сепаратори
- Операнд у изразу може да буде: константа, текућа вредност променљиве, резултат позива метода и друго.

ОПЕРАТОРИ И ИЗРАЗИ (3)

- Могу се дефинисати следећи типови израза:

```
<израз> ::= <аритметички израз>|<релациони израз>
          |<битовни израз>|<логички израз>
          |<условни израз>|<инстанцни израз>
          |<израз доделе>|<израз кастовања>
```

- Приоритет оператора одређује редослед извршавања операција приликом израчунавања (евалуације) израза

АРИТМЕТИЧКИ ОПЕРАТОРИ И ИЗРАЗИ

- Аритметички оператори, заједно са операндима и сепараторима, служе за формирање аритметичких израза
- Аритметички оператори су:

```
<аритметички оператор> ::= +|-|*|/|%|++|--
```

- Оператори + и – могу бити бинарни и унарни, префиксни и инфиксни
- Поред познатих оператора + – * и /, оператор % се користи за рачунање остатка при дељењу
- Унарни оператори ++ и –– служе за увећање, односно умањење вредности за 1 израза на који се они примењују
- Аритметички изрази служе за израчунавање бројчаних вредности

АРИТМЕТИЧКИ ОПЕРАТОРИ И ИЗРАЗИ (2)

- Аритметички израз је дефинисан на следећи начин:

```
<аритметички израз> ::= <терм>|<аритметички израз>(+|-)<терм>
<терм> ::= <фактор>|<терм>(*|/|%)<фактор>
<фактор> ::= <основни аритметички израз>|(-|+)<фактор>
<основни аритметички израз> ::= <вредност локације>[--|++]
                                |<целобројни литерал>|<реални литерал>
                                |<позив инстанцног метода>|<позив статичког метода>
                                |(<аритметички израз>)
<вредност локације> ::=<идентификатор>|<индексна променљива>|<инстанцна променљива>
                      |<референца на поље инстанце>|<референца на статичко поље>
```

- Уочава се рекурентна природа претходне дефиниције
- `<вредност локације>` представља тзв. л-вредност (енг. **l-value**) и описује све синтаксне конструкције за елементе који имају сопствену меморијску локацију
- неке металингвистичке променљиве ће бити касније описане

АРИТМЕТИЧКИ ОПЕРАТОРИ И ИЗРАЗИ (3)

- Пример. Редослед извршења операција при евалуацији аритметичког израза 7*3-7/2+4:

```
7*3 - 7/2 + 4     ► 21 - 7/2 + 4    - извршава се множење
21 - 7/2 + 4      ► 21 - 3 + 4      - извршава се целобројно дељење
21 - 3 + 4        ► 18 + 4          - извршава се одузимање
18 + 4            ► 22            - извршава се сабирање
```

БИТОВНИ ОПЕРАТОРИ И ИЗРАЗИ

- Оператор по битовима може бити логички или оператор померања.
- То су следећи оператори:

```
<битовни оператор> ::= &|||~|^|<<|>>|>>>
```

  - Прва четири међу овим операторима (&, |, ~ и ^) су логички битовни, и њима се над одговарајућим битовима операнада реализују логичке операције конјункције, дисјункције, ексклузивне дисјункције и негације
  - Последња три побројана оператора (тј. <<, >> и >>>) су оператори померања и њима се реализују операције померања за једно место над бинарним садржајем операнда: померање улево, логичко померање удесно и аритметичко померање удесно
- Битовни изрази укључују операторе по битовима. Они увек, као резултат евалуације, дају број.

БИТОВНИ ОПЕРАТОРИ И ИЗРАЗИ (2)

- Постоји велика сличност између битовних и аритметичких израза
  - разлика је једино у симболима оператора који се користе, те у њиховом приоритету и асоцијативности
- Синтакса битовног израза:

```
<битовни израз> ::= <битовни терм>|<битовни израз>(&||^)<битовни терм>
                  |~<битовни израз>
                  |<битовни израз>(<<|>>|>>>)<аритметички израз>
<битовни терм> ::= <основни битовни израз>|(&||^)<битовни терм>
<основни битовни израз> ::= <вредност локације>|<целобројни литерал>
                          |<позив инстанцног метода>|<позив класног метода>
                          |(<битовни израз>)
```

БИТОВНИ ОПЕРАТОРИ И ИЗРАЗИ (3)

- Пример. Илустује примене неких битовних оператора приликом евалуације битовних израза:

```
12 | 25      ► 29 - 00001100 | 00011001 = 00011101
12 & 25      ► 8  - 00001100 & 00011001 = 00001000
12 ^ 25      ► 21-  00001100 ^ 00011001 = 00010101
8 >> 2       ► 2 - 00001000 >> 2        = 00000010
```

ЛОГИЧКИ ОПЕРАТОРИ И ИЗРАЗИ

- Постоје три основна логичка оператора: конјункција, дисјункција и негација:

```
<логички оператор> ::= &&|||!
```

  - Оператор ! је унарни и префиксни, док су оператори && и || бинарни и инфиксни
  - Као операнди код логичких оператора могу се појављивати само подаци логичког типа

- Следеће МЛФ описују синтаксу логичког израза:

```
<логички израз> ::= <основни логички израз>
                  |<логички израз>(&&|||)<основни логички израз>
                  |!<логички израз>
<основни логички израз> ::= <идентификатор>
                          |<логички литерал>
                          |(<логички израз>)
                          |<релациони израз>
```

  - Будући да је резултат извршавања релационог израза логичка вредност, то значи да и релациони израз може учествовати у дефиницији логичког израза.

ЛОГИЧКИ ОПЕРАТОРИ И ИЗРАЗИ (2)

- Пример. Израчунавање логичког израза `(2<3)&&(3!=4)||false` се може реализовати на следећи начин:

```
(2 < 3) && (3 != 4) || false  ► true && (3!=4) || false    - прво поређење
true && (3!=4) || false       ► true && true||false        - друго поређење
true && true || false         ► true || false              - конјукција
true || false                 ► true                       - дисјункција
```

УСЛОВНИ ОПЕРАТОР И ИЗРАЗ

- Условни оператор се описује помоћу знака питања и двотачке, тј. `(? : )`
- Он се најчешће користи у у оквиру условног израза:

```
<условни израз> ::= <логички израз>?<израз>:<израз>
```

  - Извршавање условног оператора (тј. израчунавање вредности условног израза) се реализује тако што се прво одреди вредност за израз лево од упитника, тј. за мета променљиву `<логички израз>`
    - Ако је вредност тог израза `true`, тада се израчунава вредност израза између упитника и двотачке и тако добијена вредност представља коначан резулат
    - У супротном се евалуира израз десно од двотачке и та вредност представља коначан резулат

ИНСТАНЦНИ ОПЕРАТОР И ИЗРАЗ

- Помоћу инстанцног оператора `instanceof` проверава се да ли је конкретан објекат инстанца дате класе или датог интерфејса.
- Инстанцни израз има следећу форму:

```
<инстанцни израз> ::= <вредност локације> instanceof <назив класе интерфејса>
<назив класе интрфејса> ::= <идентификатор>
```

  - Металингвистичка променљива `<назив класе интерфејса>` означава назив класе, односно интерфејса, који може, али не мора, обухватати имена пакета који садрже дату класу/интерфејс
  - Оператор `instanceof` враћа вредност `true` ако је објекат, чији је назив дат са леве стране оператора `instanceof`, примерак наведене класе (или интерфејса), чије је име дато са десне стране оператора `instanceof`. У супротном враћа вредност `false`.

ОПЕРАТОР И ИЗРАЗ ЗА КРЕИРАЊЕ ОБЈЕКТА

- Помоћу оператора new креира се објекат примерак дате класе или се алоцира простор за низ.
- Израз за креирање објекта има следећу форму:

```
<израз креирања> ::= <креирање објекта>|<алокација низа>
<креирање објекта> ::= new <назив класе>([<листа аргумената>])
<назив класе> ::= <идентификатор>
<листа аргумената> ::= <аргумент>{,<аргумент>}
```

  - Променљива <назив класе> означава назив класе чија се инстанца креира - засад се сматра да је то идентификатор, мада он може, али не мора, обухватати имена пакета који садрже ту класу
  - Креирање објекта и алокација низа ће касније бити детаљно објашњени

ОПЕРАТОРИ ДОДЕЛЕ И ИЗРАЗИ ДОДЕЛЕ

- Оператор доделе, као што име казује, служи да додели вредност некој променљивој
- Оператор доделе се најчешће употребљава у следећој форми:

```
<оператор доделе> ::= =|<саставни оператор доделе>
<израз доделе> ::= <вредност локације><оператор доделе><израз>
```

  - Израз доделе се извршава тако што се евалуира израз десно од оператора доделе, а потом се израчуната вредност постави у локацију одређену вредношћу локације лево од знака једнакости
  - Израз доделе, као резултат, враћа вредност евалуираног израза десно од знака једнакости

ОПЕРАТОРИ ДОДЕЛЕ И ИЗРАЗИ ДОДЕЛЕ (2)

- Како по дефиницији израза, и израз доделе може бити израз, то се може применити рекурзивна тј. уланчана примена оператора доделе
- Пример. Следећа наредба представља уланчану наредбу доделе:

```
m = n = k = 5;
```

  - Горњом наредбом се постиже да променљива k добија вредност 5, а како је n=k, то ће променљива n добити вредност 5, и по истом принципу ће и m добити вредност 5

ОПЕРАТОРИ ДОДЕЛЕ И ИЗРАЗИ ДОДЕЛЕ (3)

- Саставни оператори доделе настају комбиновањем неких претходних оператора и простог оператора доделе:

```
<саставни оператор доделе> ::= +=|-=|*=|/=|%=|&=||=|^=|<<=|>>=|>>>=
```

  - Саставни оператор доделе служи за компактнији запис наредби увећања, умањења и слично — тако се наредба `<идентификатор>=<идентификатор><оператор><израз>` може краће записати у облику `<идентификатор><оператор>=<израз>`

- Пример. Саставни оператори доделе могу бити уланчани, као што показује следећи кôд:

```
int a=2, b=3, s=5, s1=1;
s+=s1+=a*b;
```

  - У горњем примеру променљива `s1` добија вредност 7, а променљива `s` вредност 12

ОПЕРАТОРИ: АРНОСТ, АСОЦИЈАТИВНОСТ, ПРИОРИТЕТ

- Правила су дата у табели која следи:

Приоритет	**Оператор**	**Асоцијативност**
1	(), []	неасоцијативан
2	`new`	неасоцијативан
3	.	лево-асоцијативан
4	++, --	неасоцијативан
5	-(унарни), +(унарни), !, ~, ++, --, (тип)	десно-асоцијативан
6	*, /, %	лево-асоцијативан
7	+, -	лево-асоцијативан
8	<<, >>, >>>	лево-асоцијативан

ОПЕРАТОРИ: АРНОСТ, АСОЦИЈАТИВНОСТ, ПРИОРИТЕТ

- Правила су дата у табели која следи:

Приоритет	**Оператор**	**Асоцијативност**
9	<, >, <=, >=, `instanceof`	неасоцијативан
10	==, !=	лево-асоцијативан
11	&	лево-асоцијативан
12	^	лево-асоцијативан
13	|	лево-асоцијативан
14	&&	лево-асоцијативан
15	||	лево-асоцијативан
16	?:	десно-асоцијативан
17	=, *=, /=, %=, -=, <<=, >>=, >>>=, &=, ^=, |=	десно-асоцијативан

БЕЛИНЕ

- Белина је знак који нема графички приказ на излазном уређају
- Белине служе за међусобно раздвајање елементарних конструкција и за обликовање програма
- Белине могу бити: размак, хоризонтални табулатор, знак за крај реда, знак за нову страну и знак за крај датотеке:

```
‹белина> ::= <размак>|<хоризонталини табулатор>|<знак за крај реда>
            |<знак за нову страну>|<знак за крај датотеке>
```

  - Око сваке металингвистичке променљиве у формули може бити произвољан број белина, које неће променити семанитку (значење) формуле
  - У металингвистичким формулама ће се постављати један размак између конструкција које морају бити раздвојене белинама
  - Белине у оквиру ниски утичу на семантику програма, тј. нису неутралне конструкције програмског језика Јава

КОМЕНТАРИ

- Коментари служе да се објасне поједина места у програму.
- Коментари су, пре свега, намењени човеку, али се у Јави могу искористити и за аутоматско генерисање документације
- Пожељно је на почетку програма објаснити чему програм служи, ко га је писао, када је написан итд.
- У Јави постоје 3 врсте коментара: вишелинијски (коментар у стилу језика C), једнолинијски (коментар у стилу језика C++) и документациони:

```
<коментар> ::= <једнолин. коментар>|<вишелин. коментар>|<документациони коментар>
<једнолин. коментар> ::= //{<знак не крајреда>}<знак за крај реда>
<знак не крај реда> ::= <Unicode знак различит од знака за крај реда>
<вишелин. коментар> ::= /*{{<знак не звезда>}*{<знак не слеш>}}*/
<знак не звезда> ::= <Unicode знак различит од знака ‘*’>
<знак не слеш> ::= <Unicode знак различит од знака ‘/’>
<документациони коментар> ::= /**{{<знак не звезда>}*{<знак не слеш>}}*/
```

ТИПОВИ ПОДАТАКА У ЈАВИ

- Јава је строго типизиран језик и свака операција или функција реализује се над аргументима фиксираног типа
- Тип резултата се одређује на основу типова аргумената/операнада према посебним фиксираним правилима
- Тип у Јави има следеће карактеристике.
  1. Тип података одређује скуп вредности које могу бити додељене промењљивима или изразима
  2. Над подацима из тог скупа се могу извршавати прецизно дефинисане операције, односно функције
  3. Тип израза може се одредити на основу изгледа или описа, а да није неопходно извршити његово израчунавање/евалуацију

ТИПОВИ ПОДАТАКА У ЈАВИ (2)

- Увођењем типова података омогућава се да преводилац лакше открије неисправне конструкције у језику
- Типови података доприносе прегледности програма, лакшој контроли операција од стране преводиоца и већој ефикасности преведеног програма
- У језику Јава се нови типови података дефинишу преко већ постојећих
  - Унапред морају постојати некакви примитивни (прости, предефинисани) типови података, који немају компоненте
  - Тип који **није** примитиван се при свом креирању ослања на објекте, па се овакав тип назива објектни тип.
- Како се објектима приступа преко посебних променљивих, које се називају и референце, објектни тип се још назива и референцни (или референтни) тип.

ТИПОВИ ПОДАТАКА У ЈАВИ (3)

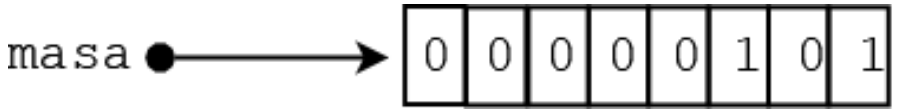
- Типови података доприносе прегледности програма, лакшој контроли операција од стране преводиоца и већој ефикасности преведеног програма
- Како се објектима приступа преко посебних променљивих, које се називају и референце, објектни тип се још назива и референцни (или референтни) тип
- Помоћу Бекусове нотације тип се дефинише на следећи начин:

```
<тип> ::= <примитивни тип>|<објектни тип>
```

ПРИМИТИВНИ ТИПОВИ ПОДАТАКА

- Ако је нека променљива примитивног типа, она представља локацију у коју ће бити смештена примитивна вредност
- Такве променљиве се још називају променљивима контејнерског типа
- Пример. Ефекат следеће наредбе:

```
byte masa = 5;
```



  - У зависности од конкретног примитивног типа, величина меморијске локације може бити различита, али она ће увек садржати вредност примитивног типа

ПРИМИТИВНИ ТИПОВИ ПОДАТАКА (2)

- Сваки примитивни тип карактерише нека кључна реч.
- Примитивни тип може бити аритметички или логички. Логички тип се описује резервисаном речју `boolean`:

```
<примитивни тип>::=<аритметички тип>|boolean
```

- Аритметички тип може бити целобројни или реални:

```
<аритметички тип> ::= <целобројни тип>|<реални тип>
```

- Постоји пет целобројних типова (у целобројни тип убраја се и знаковни):

```
<целобројни тип> ::= byte|short|int|long|char
```

- Реални тип може бити једноструке и двоструке тачности:

```
<реални тип> ::= float|double
```

ЦЕЛОБРОЈНИ ТИП ПОДАТАКА

- У оквиру целобројних типова података се разликују следећи типови: `byte`, `short`, `int`, `long` и `char`
- За сваки од тих типова постоји одређени интервал (одређен величином меморијске речи) из којег се могу узимати вредности.
  - На пример, податак типа `byte` се уписује у меморијску реч дужине један бајт (у потпуном комплементу) па су вредности из интервала $[-2^7, 2^7-1]$
  - За разлику од програмског језика C, величина примитивних типова је увек фиксирана и не зависи од платформе на којој се програм извршава. Тако су `char` и `short` увек двобајтни, `int` четворобајтни, `long` осмобајтни.
- Сви целобројни типови, осим знаковног, могу имати негативне вредности
- Цели бројеви су у Јави представљени у формату потпуног комплемента

РЕАЛНИ ТИП ПОДАТАКА

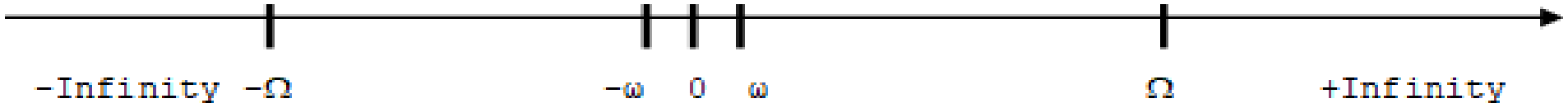
- Реални тип у Јави има за циљ представљање скупа реалних бројева у математици — што, наравно, није у потпуности могуће
- Вредности реалног типа су елементи одређени реализацијом подскупа реалних бројева на рачунару
- У програмским језицима једноставно се представљају децимални бројеви са коначно много децимала и они се третирају као приближни реални бројеви.
- У Јави постоје два реална типа: `float` и `double`, односно реални бројеви једноструке тачности и реални бројеви двоструке тачности
- У Јави се реални бројеви се интерно представљају према стандарду IEEE 754
- За представљање бројева једноструке тачности користи се бинарна реч дужине 32 бита, а за бројеве двоструке тачности реч дужине 64 бита

РЕАЛНИ ТИП ПОДАТАКА (2)

- Интервали из којих се могу представљати реални бројеви за оба типа приказани су у следећој таблици:

Тип	Интервал
float	1.40239846e-45 до 3.40282347e+38
double	4.94065645841246544e-324 до 1.79769313486231570e+308

- Реални тип можемо представити помоћу бројне осе на следећи начин:



  - Овде су са ω и Ω означени, редом, минимални и максимални реалан број по апсолутној вредности у оквиру одговарајућег реалног типа

РЕАЛНИ ТИП ПОДАТАКА (3)

- Реални бројеви из области (-∞, - Ω) не могу се регистровати и ако је резултат неке операције из тог интервала, наступило је прекорачење — тај резултат третира се као -∞ (`-Infinity`)
- Слично, бројеви из области (Ω, +∞) не могу се регистровати и третирају се као +∞ (`+Infinity`)
- Реални бројеви из области (-ω, 0)∪(0, ω) такође не могу бити регистровани. Ако је резултат неке операције из ове области, појављује се поткорачење (енг. underflow), али тај резултат се третира као нула
- Реални бројеви из области [-Ω, -ω] ∪ {0} ∪ [ω, Ω] могу се регистровати у Јави. У ствари, тачно се могу регистровати само тзв. централни бројеви, а сви остали само приближно
- Приликом оперисања са реалним бројевима може се као резултат појавити нешто што није број (на пример, ако се нула дели нулом) и стога постоји посебна вредност означена са `NaN` (енг. Not a Number).

ЗНАКОВНИ ТИП ПОДАТАКА

- Знаковни тип је одређен скупом знаковних литерала из Unicode 2.0 и операцијама над њима
- Кодна страница Unicode 2.0 се може наћи на следећој адреси: https://www.unicode.org/versions/Unicode2.0.0/
- Сваки знак у Unicode 2.0 је једнозначно одређен целим бројем, тј. својим редним бројем, па се са знаковним типом оперише као са целобројним
- Вредности овог типа су ненегативне и чувају се у меморијским локацијама величине два бајта, што значи да припадају интервалу [0, 65535]

ЛОГИЧКИ ТИП ПОДАТАКА

- Логички (`boolean`) тип је окарактерисан:
  - скупом логичких константи `true` и `false` које представљају кључне речи
  - скупом логичких оператора и операторима једнакости и неједнакости
- Логички тип је добио назив по имену енглеског математичара Була који се сматра оснивачем математичке логике
- Наредбе у Јави `if`, `while`, `for`, `do-while` и условни оператор `?:` захтевају логичке вредности за навођење услова – за разлику од неких других језика, у Јави није могућа аутоматска конверзија (кастовање) у/из логичког типа у целобројни

ОБЈЕКТНИ ТИП

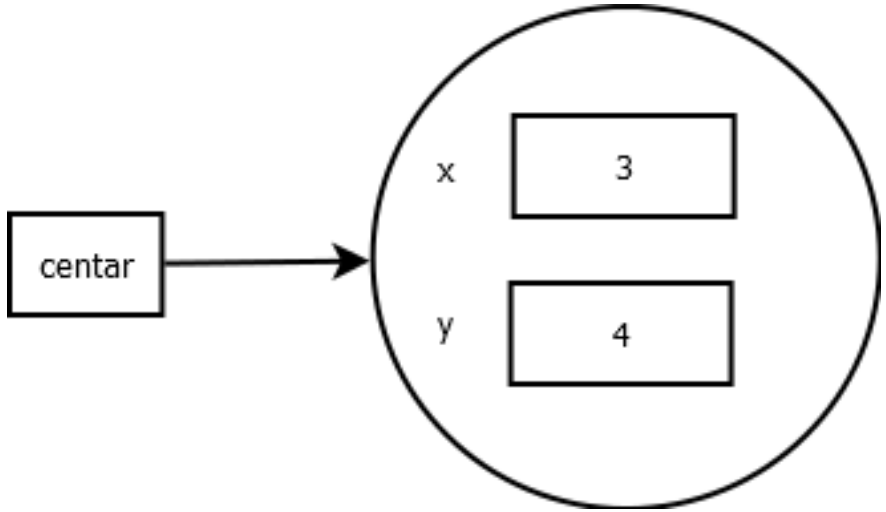
- Појам објекта је кључан у сваком објектно оријентисаном језику
- Објекат у себи обједињује скуп података и поступака за рад са тим подацима
- Објектни тип у Јави може бити:
  - кориснички — дефинише га сам корисник преко имена класе или имена интерфејса
  - низовни — може се дефинисати тако да компоненте низа буду корисничког објектног типа, или примитивног типа
  - набројиви — дефинише се преко кључне речи enum
- Горња подела може бити записана на следећи начин:

```
<објектни тип> ::= <кориснички објектни тип>|<низовни тип>|<набројиви тип>
<кориснички објектни тип> ::= <тип класе>|<тип интерфејса>
```

  - Сваки од горе наведених објектих типова ће у даљем раду бити детаљно проучен

ОБЈЕКТНИ ТИП- ОБЈЕКТИ

- Јава објекти постоје само током извршавања програма
- На нивоу меморије промењљива која представља објекат (референца) у Јави, уствари, садржи информацију којом се реферише на део меморијског простора који заузима дати објекат
- Пример. промењљива `centar` представља тачку у дводимензионалном координатном простору са координатама `(3,4)`



ОБЈЕКТНИ ТИП- КЛАСЕ

- Дефиницијом класе практично се дефинише нови тип у Јави
- Сваки примерак (или инстанца) дате класе ће имати струкутуру (атрибуте и методе) који су одређени дефиницијом класе
- Дефиниција класе би, коришћењем Бекусове нотације, могла бити дата на следећи начин:

```
<тип класе> ::= class <назив класе><тело класе>
<назив класе> ::= <идентификатор>
<тело класе> ::= <блок>
```

  - Ова дефиниција класе у није прецизна: на пример, не може сваки блок бити тело класе, већ тело класе има јасно дефинисану структуру.
  - Надаље, дефиниција није ни комплетна: засад се разматрају само класе које не користе модификаторе, наслеђивање нити имплементацију интерфејса итд
  - Како се буду уводили нови концепти, тако ће дефиниција класе бити додатно прецизирана и комплетирана.

ЕКСПЛИЦИТНА КОНВЕРЗИЈА ТИПА

- Конверзија типова у програмском језику Јава је, што се тиче примитивних типова, иста као што је то у програмском језику C – ако потреба да се ужи тип интерпретира као шири, то ће бити аутоматски урађено - програмер нема обавезу да означава да ће наступити конверзија
- Ако програмер уочи потребу да податак ширег типа буде тумачен као податак ужег типа (и на тај начин се можда изгуби прецизност или опсег), тада мора применитити експлицитну конверзију типа (кастовање) и креирати тзв. израз кастовања.
- Имплицитна и експлицитна конверзија типа постоје не само за примитивни тип, већ и за типове класа и интерфејса, при чему се информација о томе који је тип шири (општији) а који ужи, ослања на односе наслеђивања класа и на проширења и имплементације интерфејса.
- Израз кастовања има следећу синтаксу:

```
<израз кастовања> ::= (<тип>)<израз>
```

ПРОМЕНЉИВЕ

- Променљива представља локацију у меморији. Свака променљива се карактерише: именом, типом и вредношћу:
  - **име** променљиве је идентификатор
  - **тип** променљиве је један од претходно уведених типова
  - променљива садржи вредност ако је примитивног типа или референцу на објекат у супротном
- У оквиру Јава програмског језика, могу се разликовати две групе променљивих:
  - атрибути (поља)
  - локалне променљиве и аргументи метода.
- Атрибути (поља) се према животном веку могу поделити на:
  - инстанцне атрибуте (зову се и атрибути примерка или атрибути објекта)
  - класне (статичке) атрибуте

ПРОМЕНЉИВЕ (2)

- Инстанцни атрибути су везани за животни век конкретног објекта неке класе
- Класни атрибути, са друге стране, немају везе са конкретним објектима, већ са самом класом — објекат уопште не мора бити креиран да би постојао класни атрибут
- Локалне променљиве и аргументи метода се карактеришу сличним понашањем
  - Њихов животни век одговара животном веку конкретног метода који се извршава, а у ком се они налазе
  - Локалне променљиве очигледно могу бити и примитивног и референтног типа.
  - Локалне променљиве морају имати фиксирану величину, унапред познату још у фази компилације програма (јер се позиви метода ослањају на стек а да би стек могао да функционише, величина сваког оквира мора бити унапред позната.
  - Услов фиксне величине локалних променљивих је испуњен (будући да сви примитивни типови имају фиксну величину, и да се за локалне променљиве објектног типа се на стеку налазе само референце а не сами објекти)

ДЕКЛАРАЦИЈА И ИНИЦИЈАЛИЗАЦИЈА ВРЕДНОСТИ ПРОМЕНЉИВЕ

- Свака променљива мора бити декларисана, при чему се одређује и тип променљиве. Опционо, уз декларацију се може доделити и почетна вредност (иницијализација)
- Променљиве примитивних типова се декларишу и по потреби иницијализују слично као што је рађено у програмском језику C, коришћењем резервисаних речи `byte`, `short`, `int`, `long`, `char`, `boolean`, `float`, `double` за тип променљиве
- На сличан начин се декларишу и по потреби иницијализују објектне променљиве, тј. променљиве типа класе
  - Наиме, сваки објекат мора бити инстанца (примерак) неке дефинисане класе
  - Име класе које се користи за декларисање променљиве доводи до тога да декларисана променљива добија тип те класе

ДЕКЛАРАЦИЈА И ИНИЦИЈАЛИЗАЦИЈА ВРЕДНОСТИ ПРОМЕНЉИВЕ (2)

- Пример. Ако се дефинише класа `Osoba` на следећи начин:

```
class Osoba {
    ...
}
```

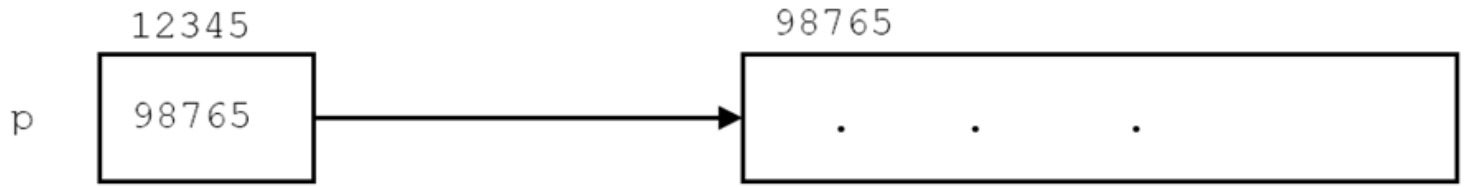
онда има смисла декларисати променљиве:

```
Osoba pera, mika;
```

Последњом Јава наредбом декларисане су две променљиве `pera` и `mika` помоћу којих се може приступати конкретним објектима, инстанцама класе `Osoba`.

ДЕКЛАРАЦИЈА И ИНИЦИЈАЛИЗАЦИЈА ВРЕДНОСТИ ПРОМЕНЉИВЕ (3)

- Начин записа података објектног типа (тј. објеката) у меморији се разликује од начина записа података примитивног типа
- У оба случаја, подацима у меморији се приступа преко променљивих:
  - Код примитивних типова променљиве садрже податке са којима се оперише
  - Код објектног типа променљиве представљају референце (показиваче) на објекте
- Пример. Ако се за променљиву p користи локација са адресом 12345, а запис објекта почиње од адресе 98765, онда се то графички представља на следећи начин:



- Могуће је декларисати више променљивих истог типа у оквиру једне вишеструке наредбе за декларацију

ДЕКЛАРАЦИЈА И ИНИЦИЈАЛИЗАЦИЈА ВРЕДНОСТИ ПРОМЕНЉИВЕ (4)

- Следеће МЛФ описују декларацију и иницијализацију променљивих:

```
<декларација и иницијализација променљивих> ::=
                <декларација и иницијализација променљивих примитивног/класног типа>
                |<декларација и иницијализација низовних променљивих>
                |<декларација и иницијализација променљивих набројивог типа>
                |<декларација променљивих типа интерфејса>
<декларација/иницијализација променљивих примитивног/класног типа> ::=
                  <назив пр./кл. типа> <листа декл./иниц. пр./кл. типа>;
<назив пр./кл. типа>::= byte|short|int|long|char|boolean|float|double|<назив класе>
<листа декл./иниц. пр./кл. типа> ::= <декл./иниц. пр./кл. типа>
                 {,<декл./иниц. пр./кл. типа>}
<декл./иниц. пр./кл. типа> ::= <назив променљиве>[=<израз>]
<назив променљиве>::= <идентификатор>
```

- У декларацији и иницијализацији најпре се наводи тип променљиве, потом следи један или више назива променљивих раздвојених зарезима
- Након назива променљиве може уследити и иницијализација

ДЕКЛАРАЦИЈА И ИНИЦИЈАЛИЗАЦИЈА ВРЕДНОСТИ ПРОМЕНЉИВЕ (5)

- Локалне променљиве морају имати постављену вредност пре него што се та вредност искористи – преводилац проверава да ли је ово задовољено тако што прилико превођења анализира изворни кôд.
- Атрибути не морају иницијализовати вредност пре коришћења пошто за њих постоје подразумеване вредности:
  - `null` уколико је реч о објектном типу,
  - `0` за нумеричке типове,
  - `'\0'` за знаковни тип,
  - `false` за логички тип.

НАРЕДБЕ

- Наредбе у програму служе за опис радњи које треба да се изврше над подацима.
- Преко извршавања наредби извршава се програм корак по корак и реализује алгоритам описан програмом
- Користимо Бекусову нотацију да дефинишемо наредбу у језику Јава:

```
<наредба> ::= <декларација и иницијализација промењљиве>|<наредба израза>
            |<блок>|<наредба гранања>|<наредба понављања>|<наредба break>
            |<наредба continue>|<обележена наредба>|<празна наредба>
            |<наредба return>|<наредба throw>|<наредба try>|<наредба synchronized>
```

  - Надаље ће бити описане све наведене наредбе, изузев прве и последње четири (прва је већ опсиана, последње четири ће бити описане касније)

НАРЕДБА ИЗРАЗА

- Наредба израза се добија тако што се на крај израза, чија је синтакса ранијеописана, дода знак **;** (тачка-зарез)

```
<наредба израза> ::= <израз>;
```

- Лако се уочава да је овим покривена и наредба доделе, јер је додела један вид израза

БЛОК

- Блок је секвенца од нула, једне или више наредби или декларација локалних променљивих ограђених витичастим заградама:

```
<блок> ::= { <наредбе блока> }
<наредбе блока> ::= {<наредба блока>}
<наредба блока> ::= <наредба>
```

- Уочава се да је претходна дефиниција блока рекурзивна:
  - блок је дефинисан преко наредбе
  - а већ је раније показано да наредба може бити блок
- Тела класа, метода итд. су блокови
- Блок може садржати друге блокове и било које друге наредбе које се извршавају једна за другом док се не наиђе на наредбу за промену тока управљања

БЛОК (2)

- У блоку могу бити декларисане локалне промењљиве
  - Оне су видљиве (могу се користити) само од места декларисања до краја блока
  - Она не може бити коришћена уколико јој није додељена почетна вредност
- Пример. Програм десно описује видљивост и блокове

```
public class TestBlok {
  public static void main(String[] args) {
    System.out.println("Здраво свете");
    {
      int x=5;
      if (x < 3) {
        int y = x++;
        System.out.println(y);
      }
      System.out.println(x);
      // System.out.println(y);
    }
    for(int x=0; x<10; x++) {
      System.out.println(x);
      // int x = 10;
    }
  }
}
```

НАРЕДБЕ ГРАНАЊА

- Наредбе гранања омогућавају доношење одлуке у зависности од испуњења неких услова
- Постоје две наредбе гранања у језику Јава: наредба `if` и наредба `switch`:

```
<наредба гранања> ::= <наредба if>|<наредба switch>
```

НАРЕДБА `if`

- Скоро сваки програмски језик омогућава неку врсту гранања у програму.
- За условно извршење наредбе или за избор између извршења две наредбе обично се користи наредба `if`
- Синтакса наредбе `if` у Јави има два облика (непотпуни и потпуни):

```
<наредба if> ::= <if-непотпуна>|<if-потпуна>
<if-непотпуна> ::= if(<израз>)<наредба>
<if-потпуна> ::= if(<израз>)<наредба>else<наредба>
```

- Наредба `if` у непотпуном облику извршава се на следећи начин:
  1. Израчунава се вредност израза
  2. Ако је вредност израза истинита, извршава се наредба која следи после израза
  3. Ако је вредност израза неистинита, извршава се прва наредба која следи после наредбе `if`

НАРЕДБА `if` (2)

- Наредба `if` у потпуном облику се извршава на следећи начин:
  1. Израчунава се вредност израза
  2. Ако је вредност израза истинита, извршава се наредба која следи после израза
  3. У супротном се извршава наредба иза резервисане речи `else`
- Пример. Наредба `if` у потпуном облику са блоковима:

```
...
if (figura.equals("krug")){
  obim = 2*pi*r;
  povrsina = pi*r*r;
}
else{
  obim = 4*a;
  povrsina = a*a;
}
...
```

НАРЕДБА `if` (3)

- Наредба условног израза (условни израз је раније описан) омогућава сажет запис потпуне наредбе `if` у програму
- Пример. Овде, уместо потпуне наредбе `if`:

```
if (x < y)
  min = x;
else
  min = y;
```

може да се наведе:

```
min = (x < y) ? x : y;
```

- У језику Јава се, код угњеждених потпуних наредби `if`, грана `else` придружује најближем `if`-у који јој претходи – при чему се блоковима може редефинисати претходно описано подразумевано понашање

НАРЕДБА `if` (4)

- Пример. Написати програм за израчунавање вредности функције $\operatorname{sgn} x = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$

при чему је *x* случајно генерисан цео број из интервала `(-10, 10)`.

Задатак решити коришћењем непотпуне наредбе `if` , те потпуне наредбе `if`.

- Пример. Написати програм за израчунавање минимума два цела броја. Бројеви се учитавају преко стандардног улаза.
- Пример. Написати програм за уређење три реална броја псеудо-случајна броја у неопадајући поредак.

НАРЕДБА switch И НАРЕДБА break

- Наредба switch служи за избор једне наредбе из скупа од неколико могућих, а на основу вредности неког израза.
- Познато је да се овај избор може извршити и помоћу наредбе if, међутим, запис помоћу наредбе switch је елегантнији и прегледнији.
- Формално, синтакса switch-наредбе се дефинише на следећи начин:

```
<наредба switch> ::= switch (<идентификатор>){
                        {case <литерал>: {<наредба>}}
                        [default:{<наредба>}]
                    }
```

НАРЕДБА `switch` И НАРЕДБА `break` (2)

- Идентификатор који следи иза резервисане речи `switch` назива се селектор и мора бити типа:
  1. `byte`, `char`, `short` или `int`
  2. енумерисани тип (од верзије Јава 5) – биће описан у касније
  3. `String`, као и типа неке од класа-омотача простих типова тј.: `Character`, `Byte`, `Short` или `Integer` (од верзије Јава 7) – биће описани касније
- Литерали (вредности) који се појављују иза резервисане речи `case` морају бити истог типа као и селектор.
- Надаље, све те вредности морају бити међусобно различите.

НАРЕДБА `switch` И НАРЕДБА `break` (3)

- Може се приметити да се приликом извршавања наредбе `switch` испитивања логичких услова спроводе секвенцијално и да се улази у придружене наредбе ако су услови испуњени
- Логички услови, за разлику од потпуне `if` наредбе, не могу имати произвољну форму, већ то увек мора бити провера једнакости селектора и вредности иза резервисане речи `case`
  - Стога је `switch`-наредба мање изражајна од потпуне `if` наредбе, односно све што се може изразити помоћу `switch` може и преко `if`, али не важи обратно
- Последњи одељак `switch`-наредбе, означен речју `default`, брине се за све преостале могућности
- Ефекат наредбе `break` је моментални излазак из целе `switch` наредбе
  - Наредба `break` се скоро увек користи, јер је захтев да вредности у оквиру `case` наредби морају бити међусобно различите (па су и логички услови који се испитују међусобно искључиви)

НАРЕДБА `switch` И НАРЕДБА `break` (4)

- Пример. Написати програм који за унету годину и редни број месеца у години исписује број дана у том месецу.

НАРЕДБЕ ПОНАВЉАЊА

- Наредбе понављања омогућавају вишеструко извршавање једне или више наредби у току једног извршавања програма.
- Наредбе чије извршавање се понавља у току једног извршавања програма образују циклус (петљу). Према томе, наредбе понављања служе за опис циклуса.
- У Јави постоје четири врсте наредби за циклусе:

```
<наредба понављања> ::= <наредба while>|<наредба do-while>|<наредба for>
                       |<колекцијска наредба for>
```

  - Прве три врсте наредби понављања су по својој форми исте као у C-у и оне ће бити одмах обрађене
  - Четврта врста, којом се описује колекцијски `for` циклус, има смисла само код низова и колекција и о њој ће бити говора приликом обраде тих тема

НАРЕДБА `while`

- Наредба `while` се још назива наредба циклуса са предусловом
- У њој се најпре проверава да ли је испуњен услов и ако јесте извршавају се наредбе циклуса
- Синтакса `while`-наредбе је следећа:

```
<наредба while> ::= while(<логички израз>)<наредба>
```

- Наредба `while` има следећи ефекат:
  1. Израчунава се вредност логичког израза
  2. Ако је вредност израза истинита, извршава се наредба и враћа се на корак 1
  3. Ако је вредност израза неистинита, завршава се извршавање наредбе `while` и прелази се на прву следећу наредбу иза наредбе `while`
- Да циклус не би био бесконачан мора постојати могућност промене вредности израза у наредби

НАРЕДБА `while` (2)

- Пример. Исписати првих 10 природних бројева применом наредбе `while`
- Пример. Написати програм за израчунавање: $S = \sum_{i=1}^{n} \frac{1}{i^3}$

НАРЕДБА do-while

- За разлику од наредбе while, услов за излазак из циклуса код наредбе do-while налази се на крају
- Ова наредба назива наредба циклуса са постусловом
- Синтакса наредбе do-while је следећа:

```
<наредба do-while> ::= do <наредба> while(<логички израз>);
```

- Извршавање наредбе do-while реализује се преко следећих корака:
  1. Извршава се наредба иза резервисане речи do
  2. Израчунава се вредност логичког израза и ако је вредност израза истинита, враћа се на корак 1
  3. У супротном, прелази се на прву наредбу иза наредбе do-while

НАРЕДБА `do-while` (2)

- Све што се описује помоћу наредбе `do-while` може се, такође, описати помоћу наредбе `while`. Наиме, наредбом:

```
do <наредба> while(<логички израз>)
```

постиже се исти ефекат као и наредбама:

```
<naredba>
while (<логички израз>) <наредба>
```

- Пример. Написати програм који генерише и исписује случајне бројеве све док се не добије број већи или једнак `0.9`
- Пример. Написати програм за израчунавање дужине (броја цифара) унетог целог броја типа `long`
- Пример. Написати програм за уношење низа речи (једна реч у једном реду) са стандардног улаза све док се не препозна реч „КРАЈ“. Затим одштампати укупан број речи као и број појављивања речи: „програмирање“, „математика“ „физика“

НАРЕДБА for - БРОЈАЧКИ ЦИКЛУС

- Наредба for је моћна наредба за опис циклуса
- Најчешће се користи за опис циклуса код којих је број понављања наредби унапред познат и у том смислу она представља типичну „бројачку” наредбу
- Наредба for се може третирати као боље структурирана наредба понављања у поређењу са наредбом while и може се користити свуда где се користи и наредба while
- Синтакса наредбе for је следећа:

```
<наредба for> ::= for([<иницијализација>];[<услов>];[<итерација>])<наредба>
<иницијализација> ::= <листа израза>|<декларација локалне променљиве>
<услов> ::= <логички израз>
<итерација> ::= <листа израза>
<листа израза> ::= <израз>{,<израз>}
```

НАРЕДБА `for` - БРОЈАЧКИ ЦИКЛУС (2)

- Наредба `for` се извршава на следећи начин:
  1. Најпре се извршава иницијализациони део петље `for`. Ако нема садржаја у овом делу, не врши се иницијализација
  2. Након иницијализације, проверава се да ли је услов останка у петљи (ако постоји) испуњен. Ако услов није дат, подразумева се да је услов испуњен.
  3. Ако је услов испуњен тј. логички израз има вредност `true`, извршава се наредба петље (тело петље), а затим се извршава итерација петље, па се поново се иде на корак 2
  4. Уколико услов није испуњен, тј. израз има вредност `false`, окончава се извршавање петље
- У наредби `for` често се појављује променљива помоћу које се врши пребројавање и која се назива бројач
  - Пребројавање може да се врши уз увећање вредности бројача (бројање унапред), почев од неке почетне вредности, све док је услов испуњен тј. тачан
  - Алтернативно, пребројавање може да се врши уз умањење вредности бројача (бројање уназад)), почев од неке почетне вредности, све док је услов испуњен

НАРЕДБА for - БРОЈАЧКИ ЦИКЛУС (3)

- Пример. Написати програм који израчунава факторијел унетог броја применом циклуса for
- Пример. Написати програм за израчунавање суме првих n природних бројева користећи for наредбу за бројање уназад
- Пример. Реализовати бесконачи for циклус
- Пример. Написати програм којим се израчунава сваки наредни степен променљиве (чија почетна вредност је већа од 0, а мања од 1) све док не постане мањи или једнак 0.1
- Пример. Написати програм који за аргумент из интервала [0, 1] са кораком 0.1 рачуна и приказује вредност корена броја
- Пример. Написати програм који исписује индексе елемената на споредној дијагонали квадратне матрице. Димензија матрице је задата природним бројем

НАРЕДБА `break` И ЦИКЛУСИ

- Раније приказана је употреба наредбе `break` у контексту `switch` наредбе
- Употреба у контексту циклуса је слична (може се користити уз било који тип циклуса) и ефекат је прекид рада циклуса
- Битно је напоменути да се прекид односи само на најближи циклус, односно онај ком `break` наредба директно припада
  - Дакле, у случају угњежденог циклуса се `break` унутар унутрашњег циклуса односи на прекид рада унутрашњег, али не и спољашњег циклуса.
- Синтакса наредбе `break` је следећа:

```
<наредба break> ::= break[ <обележје>];
<обележје> ::= <идентификатор>
```

  - Као што се може видети, постоји могућност да нареба `break` буде са обележјем, тј. да се иза кључне речи `break` нађе обележје

НАРЕДБА `break` И ЦИКЛУСИ (2)

- Пример. Помоћу бесконачног `while` циклуса и `break` наредбе, написати програм који сумира све унете бројеве док се не унесе први негативан број. Након уноса негативног броја извршавање програма се прекида.

НАРЕДБА continue И ЦИКЛУСИ

- Наредба continue, за разлику од break, не прекида придружени циклус већ наставља са његовом наредном итерацијом, а прескаче сав кôд који се налази до краја актуелне итерације
- Синтакса наредбе continue је следећа:

```
<наредба continue> ::= continue[ <обележје>];
<обележје> ::= <идентификатор>
```

  - Као што се може видети, постоји могућност и да нареба continue буде са обележјем, што ће бити обрађено у следећој секцији
- Пример. Помоћу бесконачног while циклуса, continue и break наредби написати програм који сумира све позитивне бројеве док се не унесе број 0, након чега се извршавање програма прекида

ОБЕЛЕЖЕНА НАРЕДБА

- Наредба у програмском језику може имати једно или више обележја (ознака) и онда се назива обележеном наредбом
- Синтакса обележене наредбе је следећа:

```
<обележена наредба> ::= {<обележје>:}<наредба>
<обележје> ::= <идентификатор>
```

- Обележена наредба има ограничену употребу у језику Јава и користи се у комбинацији са наредбама `break` и `continue` када се жели безусловни пренос управљања на одређено место у програму
- Обележене наредбе највише смисла имају у оквиру угњеждених циклуса
- Пример. Написати програм који проверава да ли се неки задати текст налази као под-текст (подниска) другог задатог текста

ПРАЗНА НАРЕДБА

- Празна наредба је наредба без дејства.
- Користи се у оним деловима програма где нема никаквих акција
- Њена синтакса се може овако изразити:

```
<празна наредба> ::= ;
```

- Пример. У следећој наредби `for` тело петље је празна наредба:

```
for (x=a; x<81; x+=5)
    ;
```

ПИТАЊА И ЗАДАЦИ

- Шта је основна разлика између природних (говорних) језика и програмских језика?
- Шта су граматика, синтакса и семантика језика?
- Дефинисати реалне бројеве у модификованој Бекусовој нотацији.
- Да ли су следеће речи идентификатори? Образложити одговор.
  - new
  - a1
  - prvi Broj
  - 8$4
  - prvi_Broj
  - finally
  - $x#
  - true
- Који од следећих литерала су исправно записани? Образложити одговор.
  - "abc " def"
  - '\x'
  - '0'
  - 23.14-56
  - 23.14E5
  - 0X45G

ПИТАЊА И ЗАДАЦИ (2)

- Да ли има разлике у префиксној и постфиксној примени оператора ++ и --? Илустровати примерима.
- Нека су дати:

```
int a = 4, b = 3, c = 0, d = 2;
boolean uslovA, uslovB;
uslovA = uslovB = ((a—)>=(++b)) || ((++c)==(3/d));
```

- Колике су вредности променљивих a, b, c, d, uslovA и uslovB након извршења наведног дела кода?
- Нека је дат 8-битни запис целог неозначеног броја 00010101. Извршити померање за једну позицију улево и објаснити како померање утиче на декадну вредност датог броја. Након тога, извршити померање броја за једну позицију удесно и објаснити како померање утиче на декадну вредност датог броја.
- За шта се користе коментари? Које врсте коментара постоје у програмском језику Јава?
- Објаснити шта значи да је Јава строго типизиран језик. Истражити који су још програмски језици строго типизирани, као и шта је разлика између строго и слабо типизираних језика?

ПИТАЊА И ЗАДАЦИ (3)

- Примерима илустровати ситуацију у којој постоји разлика између употребе do-while наредбе понављања и употребе while или for наредби понављања.
- Објаснити шта су обележене наредбе и примером илустровати ситуацију када је корисно да се користе.
- Шта је празна наредба и у којим ситуацијама се користи?