



# Akademija tehničko-vaspitačkih studija odsek NIŠ

Katedra za Informaciono-komunikacione tehnologije

## OBJEKTNO ORIJENTISANO PROGRAMIRANJE - OOP

Prof. dr Zoran Veličković, dipl. inž. el.

2019/2020.

Prof. dr Zoran Veličković, dipl. inž. el.

# OBJEKTNO ORIJENTISANO PROGRAMIRANJE - OOP

---

Paketi i tipovi podataka u JAVI

(4)

# Sadržaj

## ➤ PROSTOR IMENA U JAVI

- Javin hijerarhijski prostor imena
- Paket kao kontejner klasa
- Uvoženje paketa
- Metode za formatirano štamanje

## ➤ TIPOVI PODATAKA U JAVI

- Elementarni – ugrađeni tipovi
- Referencni tipovi
- Promenljive u Javi

## ➤ NIZOVI U JAVI

- Java: 1D/2D nizovi
- Java: xD nizovi
- Znakovni nizovi u Javi

## ➤ UPRAVLJANJE PROGRAMSKIM IZVRŠENJEM

- Operatori poređenja
- Blok koda u Javi
- Oblast važenja u Javi



# Prostori imena u Javi - paketi

- ▶ U Javi se definišu **PROSTORI IMENA** u kojima se smeštaju **METODE** i **PODACI** koji se koriste u odgovarajućoj klasi.
- ▶ Ovo omogućava da više metode mogu **IMATI ISTA IMENA** ako pripadaju **RAZLIČITIM** imenskim prostorima.
- ▶ Prostori imena u Javi je **HIJERARHIJSKI** organizovan slično strukturi foldera odvojenih tačkom.
- ▶ Zbog toga, da bi se definisalo **KOJA METODA** se tačno poziva, **MORA** se navesti **PROSTOR IMENA**.
- ▶ U Javi se **PROSTOR IMENA** naziva **PAKET**.
- ▶ Primer: umesto:

piše se:

`println("Hello World!");`

Poziv metode `println()` iz podrazumeanog paketa

`System.out.println("Hello World!");`

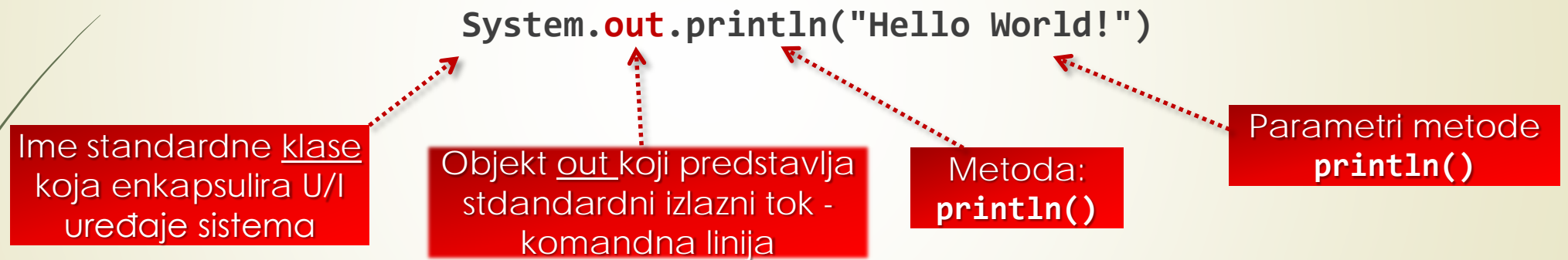
Poziv metode `println()` iz paketa `System.out`

# Hijerarhijski organizovani paketi u Javi

- ▶ Tako se umesto:

```
println("Hello World!"),
```

može (i treba) pisati:



- ▶ Za **SMEŠANJE KLASA** i **PODELU IMENSKOG PROSTORA** u Javi se koriste **PAKETI**.
- ▶ Da bi se definisalo **KOJA** se metoda tačno poziva, mora se koristiti **HIJERARHIJSKI** pristup metodi određen preko **IMENA PAKETA** i **KLASE**.
- ▶ Klasa **System** se nalazi u paketu **java.lang** koji se **PODRAZUMEVANO UČITAVA** u **JAVA** projekte i nije je potrebno eksplicitno uključivati u projekte.

# Oracle paketi: java.lang - **println**

The screenshot shows the Oracle Java API documentation for the `println` method in the `java.lang` package. The browser address bar shows the URL: <https://docs.oracle.com/javase/8/docs/api/index.html?java/lang/Class.html>. The left sidebar lists various Java packages, with `java.lang` highlighted. The main content area shows the `Field Detail` for the `out` field, which is a `PrintStream`. The description explains that this is the standard output stream and provides an example of how to use it: `System.out.println(data)`. The `err` field is also partially visible at the bottom.

**Svi Java paketi** →

**paket** → `java.lang`

**klasa** → `System`

**Field Detail**

**in**

`public static final InputStream in`

The "standard" input stream. This stream is already open and ready to supply input data. Typically this stream corresponds to keyboard input or another input source specified by the host environment or user.

**out**

`public static final PrintStream out`

The "standard" output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user.

For simple stand-alone Java applications, a typical way to write a line of output data is:

→ `System.out.println(data)`

See the `println` methods in class `PrintStream`.

**See Also:**

`PrintStream.println()`, `PrintStream.println(boolean)`, `PrintStream.println(char)`, `PrintStream.println(char[])`, `PrintStream.println(double)`, `PrintStream.println(float)`, `PrintStream.println(int)`, `PrintStream.println(long)`, `PrintStream.println(java.lang.Object)`, `PrintStream.println(java.lang.String)`

**err**

# Metoda za formatirano štampanje `println`

- ▶ U `println()` metodi se mogu koristiti i tzv. **ESCAPE SEKVENCE** (to su sve sekvence koje počinju „backslash“ karakterom “\”) za **FORMATIRANO ŠTAMPANJE** na komandnoj liniji.
- ▶ Ove sekvence su nasleđene iz ere **LINIJSKIH ŠTAMPAČA** i **TELEPRINTERA!**
- ▶ Tabela najčešće korišćenih “**ESCAPE**” karaktera:

| Char            | ENGL. IME       | OBJAŠNJENJE   |
|-----------------|-----------------|---|
| <code>\n</code> | Newline         | Postavi poziciju kursora na početak sledeće linije.   |
| <code>\t</code> | Horizontal tab  | Pomeri kursor do sledeće tabulir pozicije.  |
| <code>\r</code> | Carriage return | Postavi kursor na početak tekuće linije, ne prelazi se u sledeću liniju. Prethodno ispisani karakteri će biti prepisani novim.          |
| <code>\\</code> | Backslash       | Koristi se za štampanje “backslash” karaktera.  |
| <code>\"</code> | Double quote    | Koristi se za štampanje “double-quote” karaktera. Primer:<br><code>System.out.println( "\"in quotes\"" );</code> prikazuje "in quotes". |

# Oracle paketi: java.lang – printf

The screenshot displays the Oracle Java API documentation for the `printf` method. The browser address bar shows the URL: <https://docs.oracle.com/javase/8/docs/api/index.html?java/lang/Class.html>. The left sidebar shows the package hierarchy, with `java.io` and `PrintStream` highlighted. The main content area shows the following details:

```
public PrintStream printf(String format,
                          Object... args)
```

A convenience method to write a formatted string to this output stream using the specified format string and arguments.

An invocation of this method of the form `out.printf(format, args)` behaves in exactly the same way as the invocation

```
out.format(format, args)
```

**Parameters:**

- `format` - A format string as described in [Format string syntax](#)
- `args` - Arguments referenced by the format specifiers in the format string. If there are more arguments than format specifiers, the extra arguments are ignored. The number of arguments is variable and may be zero. The maximum number of arguments is limited by the maximum dimension of a Java array as defined by *The Java™ Virtual Machine Specification*. The behaviour on a null argument depends on the [conversion](#).

**Returns:**

This output stream

**Throws:**

- `IllegalFormatException` - If a format string contains an illegal syntax, a format specifier that is incompatible with the given arguments, insufficient arguments given the format string, or other illegal conditions. For specification of all possible formatting errors, see the [Details](#) section of the formatter class specification.
- `NullPointerException` - If the format is null

**Since:**

1.5



# Metode za formatirano štampanje `printf`

```
System.out.println("Welcome\n\tto\n\tJava\n\tProgramming!");
```

- ▶ Za **FORMATIRANO ŠTAMPANJE** može se koristiti i metoda `printf()` - poznata iz C-a!
- ▶ Metoda `printf()` koristi **ZAPETAMA ODVOJENU LISTU** koja može sadržavati **VIŠE ARGUMENATA** za štampanje.
- ▶ Primer formatiranog štampanja **TEKSTA** - stringa (`%s`) u **DVE LINIJE**:

```
System.out.printf("%s\n%s\n", "Dobrodošli u", "programiranje na Javi!");
```

- ▶ Evo primera štampanja decimalnih (`%d`) **CELIH BROJEVA**:

```
System.out.printf("Zbir je %d\n", zbir);
```

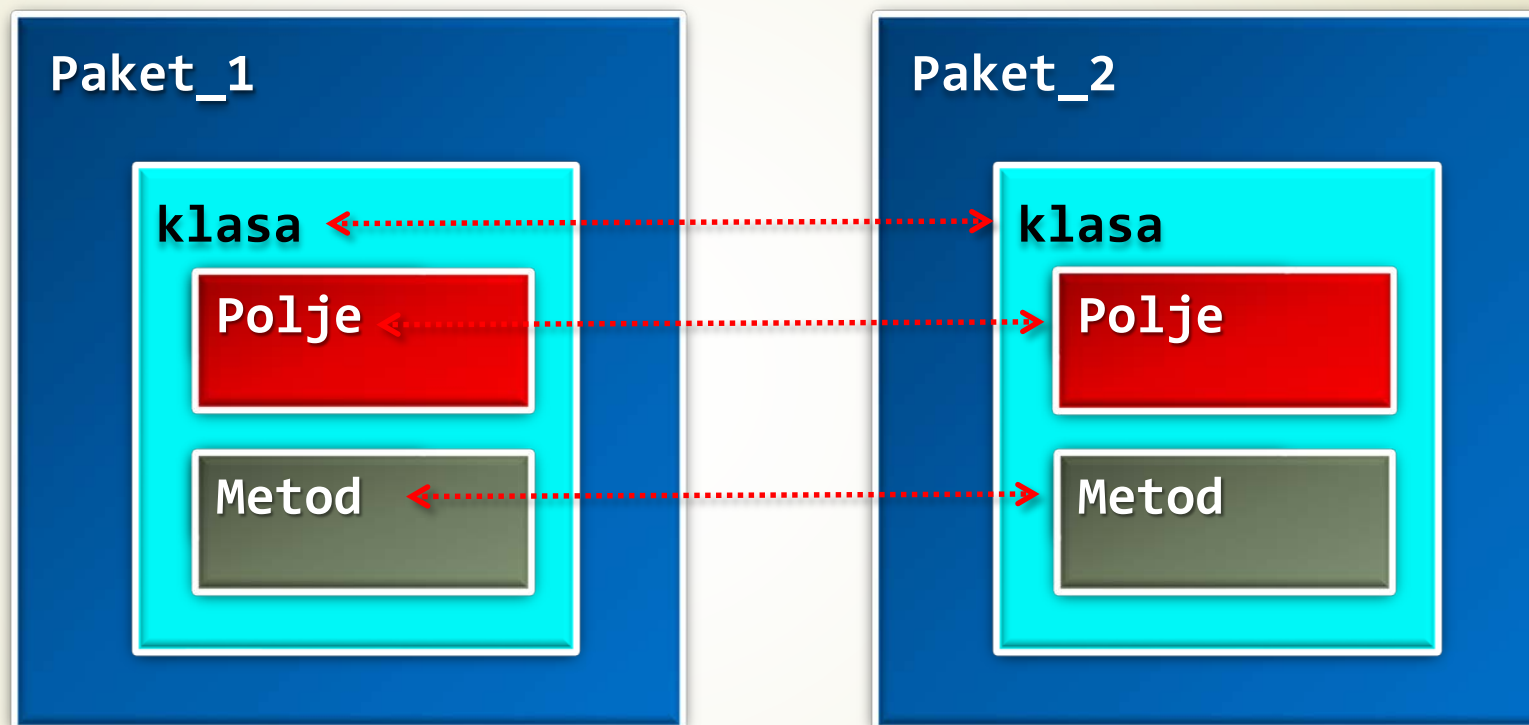
- ▶ Za **UČITAVANJE CELIH DECIMALNIH BROJEVA** sa **TASTATURE** (standardnog ulaza) može se koristiti metoda `nextInt()` na sledeći način:

```
Int_Broj=input.nextInt();
```

# Paket kao kontejner klasa

- ▶ Već znamo, za smešanje **KLASA** i **PODELU** imenskog prostora u Javi koriste se **PAKETI**.
- ▶ **PAKETI** predstavljaju **SKUP KLASA** (kaže se kontejner) sa **JEDINSTVENIM IMENOM**.
- ▶ **SVAKA KLASA** u Javi je **DEO** nekog **PAKETA!**
- ▶ Paketi se čuvaju **HIJERARHIJSKI**, a **IZRIČITO SE UVOZE**, u definiciju **NOVE KLASE** (postoji jedan izuzetak – paket **Java.lang**).
- ▶ Novi paketi se **FORMIRAJU** komandom "**package**" koja mora biti **PRVA NAREDBA** u izvornoj datoteci.
- ▶ Za hijerarhijsko čuvanje paketa koristi se **SISTEM DIREKTORIJUMA**.
- ▶ Tako, datoteke sa ekstenzijom **.class** za bilo koju klasu deklarisanu u paketu **MojPaket** **MORA** da bude u direktorijumu **MojPaket!**

# Paketi



Dva **NEZAVISNA PAKETA** Paket\_1 i Paket\_2 sa svojim metodama i poljima MOGU imati **ISTA IMENA** a da pri kompajliranju **NE DOĐE** do problema.

# Uvoženje paketa (1)

- ▶ Paketi se organizuju u hijerarhiju primenom OPERATORA TAČKA (.), primer:

```
package java.awt.image;
```

- ▶ Tako, prethodni primer zahteva da paket bude smešten u direktorijum:

```
java\awt\image
```

- ▶ **UVOŽENJE** već formiranog paketa u programski kod se obavlja rezervisanom reči "**import**" na sledeći način:

```
import java.util.Date; ili import java.lang.*;
```

- ▶ Zvezdicom (\*) se označava uvoženje **CELOG PAKETA** (svih klasa i promenljivih).
- ▶ Iako uvoženje čitavog paketa produžava vreme kompajliranja, to **NEMA NEGATIVNOG EFEKTA** u trenutku **IZVRŠAVANJA** programa!

## Uvoženje paketa (2)

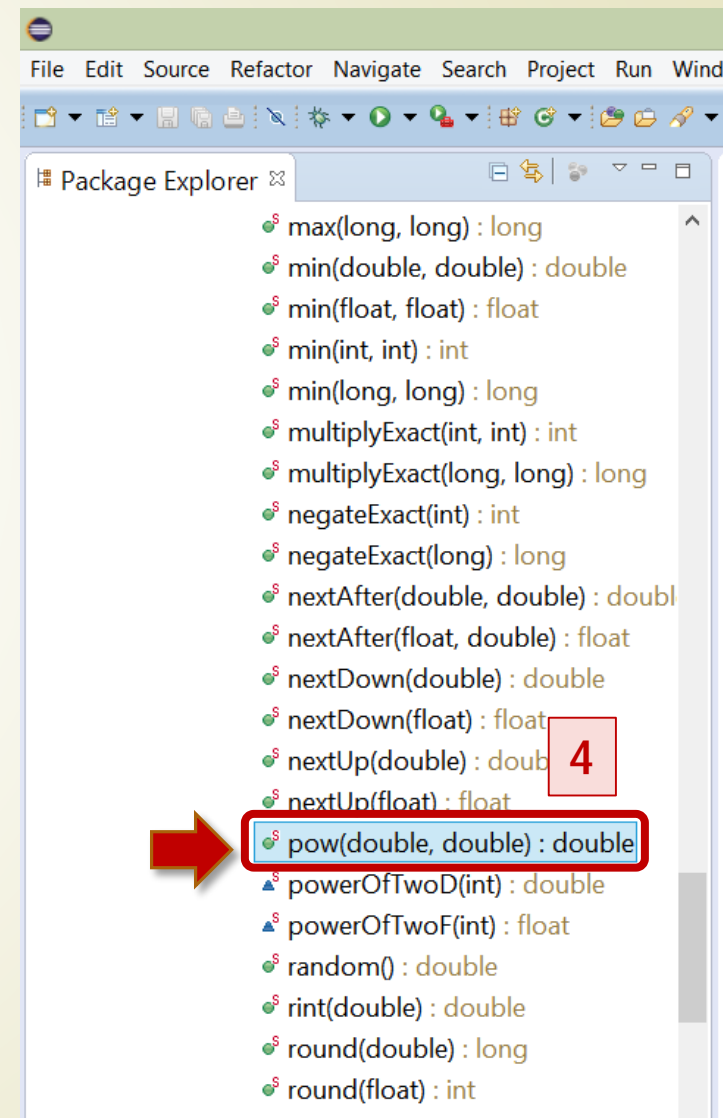
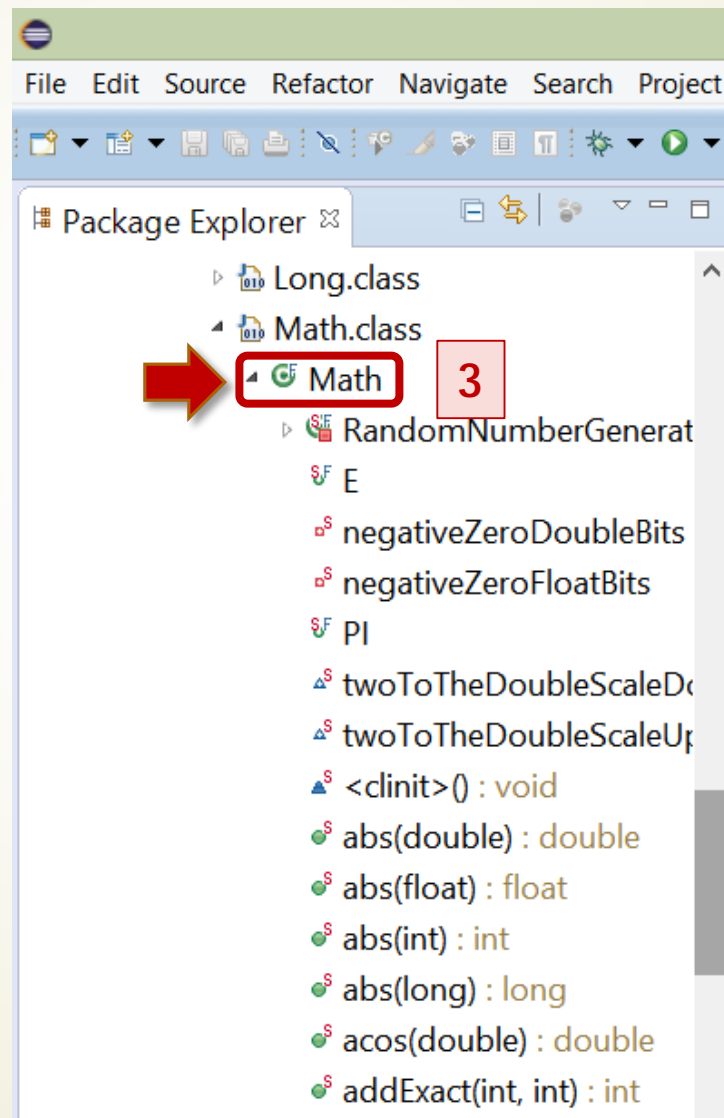
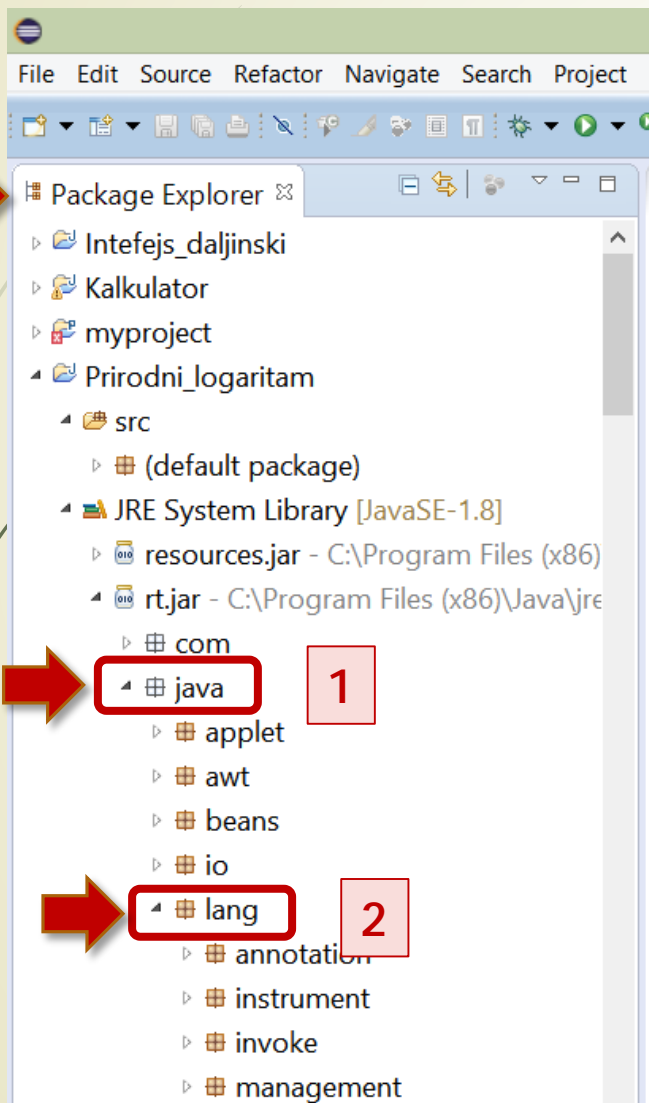
- ▶ Standardne (ugrađene) Javine klase koje su neophodne u kreiranju i najjednostavnijih programa se čuvaju u **PAKETU** pod nazivom **java**.
- ▶ **OSNOVNE FUNKCIJE** samog **JEZIKA** se čuvaju u potpaketu java.lang koji se **AUTOMATSKI UVOZI** u **SVE** programe.
- ▶ OO paradigma koja se odnosi na **KAPSULIRANJE i BEZBEDNOST IZVRŠAVANJA** programskog koda je u Javi delom **REALIZOVANA PAKETIMA**.
- ▶ Za definisanje prava pristupa u Javi su predviđene ključne reči: **protected, public, private, ...**
- ▶ Međutim, **PRAVO PRISTUPA** klasama i metodama se definiše i **PAKETIMA!**
- ▶ U zavisnosti od konteksta, kreiran se prilično **SLOŽEN SISTEM** prava pristupa pojedinim klasama i promenljivama.

## Uvoženje paketa (2)

| PAKET              | OBJAŠNJENJE   |
|--------------------|---|
| <b>java.lang</b>   | Sadrži klase koje su od fundamentalnog značaja za Javu (npr. klasa Math). Sve promenljive su dostupne u svojim programima automatski. Nema potrebe za naredbom uvoza ovog paketa.       |
| <b>java.io</b>     | Sadrži klase koje se odnose na podršku ulazno/izlaznim operacijama (tokovima).  |
| <b>java.awt</b>    | Sadrži klase koje podržavaju Javin grafički korisnički interfejs (GUI). Iako se ove klase mogu koristiti za GUI programiranje, preporučuje se korišćenje alternativne klase Swing.      |
| <b>javax.swing</b> | Obezbeđuje klase koje podržavaju "Swing" GUI komponente. One nisu samo fleksibilnije i lakše za korišćenje u odnosu na java.awt ekvivalente, već su minimalno zavisne od izvornog koda. |
| <b>java.applet</b> | Sadrži klase koje podržavaju pisanje apleta – programa integrisanih u Web stranicu (obrađivano u predmetu Internet tehnologije).  |
| <b>java.util</b>   | Sadrži klase za široku podršku standardnim operacijama kolekcija, vremenskim informacijama i stringovima.   |

# Hijerarhija Javinih paketa u Eclipse-u

Java paketi




# Skoro svi Javini paketi!

java.applet, java.awt, java.awt.color, java.awt.datatransfer, java.awt.dnd, java.awt.event, java.awt.font, java.awt.geom, java.awt.im, java.awt.im.spi, java.awt.image, java.awt.image.renderable, java.awt.print, java.beans, java.beans.beancontext, java.io, java.lang, java.lang.annotation, java.lang.instrument, java.lang.management, java.lang.ref, java.lang.reflect, java.math, java.net, java.nio, java.nio.channels, java.nio.channels.spi, java.nio.charset, java.nio.charset.spi, java.rmi, java.rmi.activation, java.rmi.dgc, java.rmi.registry, java.rmi.server, java.security, java.security.acl, java.security.cert, java.security.interfaces, java.security.spec, java.sql, java.text, java.text.spi, java.util, java.util.concurrent, java.util.concurrent.atomic, java.util.concurrent.locks, java.util.jar, java.util.logging, java.util.prefs, java.util.regex, java.util.spi, java.util.zip, javax.accessibility, javax.activation, javax.activity, javax.annotation, javax.annotation.processing, javax.crypto, javax.crypto.interfaces, javax.crypto.spec, javax.imageio, javax.imageio.event, javax.imageio.metadata, javax.imageio.plugins.bmp, javax.imageio.plugins.jpeg, javax.imageio.spi, javax.imageio.stream, javax.jws, javax.jws.soap, javax.lang.model, javax.lang.model.element, javax.lang.model.type, javax.lang.model.util, javax.management, javax.management.loading, javax.management.modelmbean, javax.management.monitor, javax.management.openmbean, javax.management.relation, javax.management.remote, javax.management.remote.rmi, javax.management.timer, javax.naming, javax.naming.directory, javax.naming.event, javax.naming.ldap, javax.naming.spi, javax.net, javax.net.ssl, javax.print, javax.print.attribute, javax.print.attribute.standard, javax.print.event, javax.rmi, javax.rmi.CORBA, javax.rmi.ssl, javax.script, javax.security.auth, javax.security.auth.callback, javax.security.auth.kerberos, javax.security.auth.login, javax.security.auth.spi, javax.security.auth.x500, javax.security.cert, javax.security.sasl, javax.sound.midi, javax.sound.midi.spi, javax.sound.sampled, javax.sound.sampled.spi, javax.sql, javax.sql.rowset, javax.sql.rowset.serial, javax.sql.rowset.spi, javax.swing, javax.swing.border, javax.swing.colorchooser, javax.swing.event, javax.swing.filechooser, javax.swing.plaf, javax.swing.plaf.basic, javax.swing.plaf.metal, javax.swing.plaf.multi, javax.swing.plaf.synth, javax.swing.table, javax.swing.text, javax.swing.text.html, javax.swing.text.html.parser, javax.swing.text.rtf, javax.swing.tree, javax.swing.undo, javax.tools, javax.transaction, javax.transaction.xa, javax.xml, javax.xml.bind, javax.xml.bind.annotation, javax.xml.bind.annotation.adapters, javax.xml.bind.attachment, javax.xml.bind.helpers, javax.xml.bind.util, javax.xml.crypto, javax.xml.crypto.dom, javax.xml.crypto.dsig, javax.xml.crypto.dsig.dom, javax.xml.crypto.dsig.keyinfo, javax.xml.crypto.dsig.spec, javax.xml.datatype, javax.xml.namespace, javax.xml.parsers, javax.xml.soap, javax.xml.stream, javax.xml.stream.events, javax.xml.stream.util, javax.xml.transform, javax.xml.transform.dom, javax.xml.transform.sax, javax.xml.transform.stax, javax.xml.transform.stream, javax.xml.validation, javax.xml.ws, javax.xml.ws.handler, javax.xml.ws.handler.soap, javax.xml.ws.http, javax.xml.ws.soap, javax.xml.ws.spi, javax.xml.ws.wsaddressing, javax.xml.xpath, org.ietf.jgss, org.omg.CORBA, org.omg.CORBA\_2\_3, org.omg.CORBA\_2\_3.portable, org.omg.CORBA.DynAnyPackage, org.omg.CORBA.ORBPackage, org.omg.CORBA.portable, org.omg.CORBA.TypeCodePackage, org.omg.CosNaming, org.omg.CosNaming.NamingContextExtPackage, org.omg.CosNaming.NamingContextPackage, org.omg.Dynamic, org.omg.DynamicAny, org.omg.DynamicAny.DynAnyFactoryPackage, org.omg.DynamicAny.DynAnyPackage, org.omg.IOP, org.omg.IOP.CodecFactoryPackage, org.omg.IOP.CodecPackage, org.omg.Messaging, org.omg.PortableInterceptor, org.omg.PortableInterceptor.ORBInitInfoPackage, org.omg.PortableServer, org.omg.PortableServer.CurrentPackage, org.omg.PortableServer.POAManagerPackage, org.omg.PortableServer.POAPackage, org.omg.PortableServer.portable, org.omg.PortableServer.ServantLocatorPackage, org.omg.SendingContext, org.omg.stub.java.rmi, org.w3c.dom, org.w3c.dom.bootstrap, org.w3c.dom.events, org.w3c.dom.ls, org.xml.sax, org.xml.sax.ext, org.xml.sax.helpers



# Tipovi podataka u Javi (1)


- ▶ Java je strogo **TIPIZIRAN JEZIK** (šta to zapravo znači?) - svaka promenljiva i svaki izraz u Javi moraju da **IMAJU SVOJ TIP** (inače će kompajler prijaviti grešku)!
- ▶ **APSTRAKCIJE PODATAKA** omogućava željeni nivo bezbednosti, je se pri dodeljivanju vrednosti **PROMENLJIVAMA** ili **IZRAZIMA** mogu uzeti samo unapred **PREDEFINISANE VREDNOSTI**.
- ▶ Java vrši **STROGU PROVERU TIPa** i programski kod čini bezbednijim. 
- ▶ U Javu su ugrađeni **ELEMENTARNI** (ili vrednosni, odnosno, prosti) tipovi podataka kao što su: **bajt, short, int, long, char, float, double** i **boolean**.
- ▶ Svi **ELEMENTARNI TIPOVI** su svrstani u sledeće **GRUPE**:
  - ▶ celi brojevi,
  - ▶ brojevi u pokretnom zarezu,
  - ▶ znakovi i
  - ▶ logičke vrednosti.

## Tipovi podataka u Javi (2)

- ▶ Setite se da je u Javi sve u **KLASAMA**, međutim, prosti tipovi podataka **NISU OBJEKTI** (odmah pa izuzetak!).
- ▶ Razlog ovome leži u **EFIKASNOSTI** predstavljanja i korišćenje podataka u memoriji.
- ▶ Programer **NE MOŽE** kreirati prikazane **ELEMENTARNE TIPOVE**, već samo može **IZABRATI** odgovarajući (u zavisnosti od aplikacije).
- ▶ Ako se želi dodeljivanje vrednosti promenljivoj iz **VEĆEG BROJNOG OPSEGA** (engl. *range*) onda je neophodno rezervisati i veću količinu memorije (bilo **VIŠE BITOVA** ili **BAJTOVA**).

# Elementarni tipovi u Javi

| TIP PODATAKA | VELIČINA [BAJT] | OPSEG  |
|--------------|-----------------|--|
| bajt         | 1               | -128 do +127                                   |
| boolean      | 1               | Tačno ili netačno                              |
| char         | 2               | A-Z, a-z, 0-9, spec char.                      |
| short        | 2               | -32768 do +32767                               |
| int          | 4               | -2 miliona do +2 miliona                       |
| long         | 8               | $-10^{18}$ do $+10^{18}$                       |
| float        | 4               | $-3.4 \cdot 10^{38}$ do $+3.4 \cdot 10^{38}$   |
| double       | 8               | $-1.7 \cdot 10^{308}$ do $+1.7 \cdot 10^{308}$ |



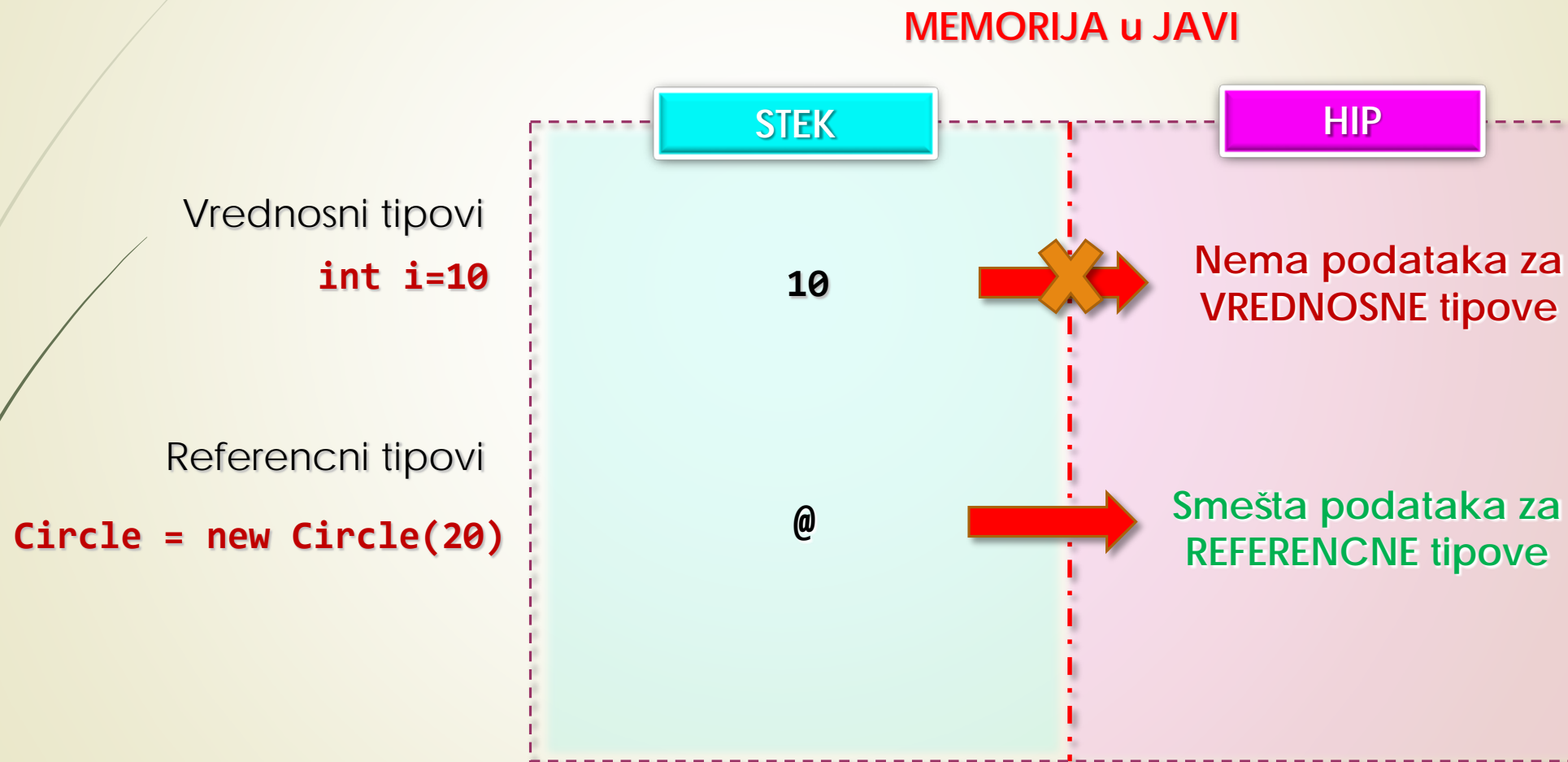
# Vrednosni tipovi u Javi

- ▶ Šta su zapravo **TIPOVI PODATKA**?
- ▶ **TIPOVI PODATKA** se mogu smatrati **ŠABLONIMA** koji opisuju **FUNKCIONALNOST** kolekcije podataka.
- ▶ Pored **VREDNOSNIH TIPOVA** podataka, u Javi postoje i **REFERENCNI TIPOVI**.
- ▶ Osnovna razlika između ovih tipova je u **NAČINU SMEŠTANJA** (u memoriji) i **PRISTUPA** podacima.
- ▶ Generalno, **MEMORIJA ZA PODATKE** je u Javi podeljena na **STEK** i **HIP** (engl. *heap*) deo.
- ▶ **VREDNOSNI** (elementarni) tipovi podataka se smeštaju u **STEK DEO MEMORIJE** (**integer**, **boolean**, **char**, **struct**).
- ▶ Posle korišćenja, ovi tipovi podataka **OSLOBAĐAJU** zauzetu memoriju.
- ▶ Ovaj način korišćenja memorije je bio prisutan i u **STARIJIM PROGRAMSKIM JEZICIMA**.

# Referencni tipovi u Javi

- ▶ Za razliku od vrednosnih tipova, promenljive **REFERENCNOG TIP** postoje **ISTOVREMENO** u **OBA MEMORIJSKA DELA** (dakle, i u hipu i na steku).
- ▶ Konkretni podaci **OBJEKTA** se nalaze u **DINAMIČKOM DELU MEMORIJE - HIP-u**.
- ▶ **ISTOVREMENO** se formira promenljiva (naziva se **REFERENCA** te otuda i naziv) na **STEKU** koja ukazuje na objekt u **HIP** memoriji.
- ▶ Kada neka funkcija pristupa referencnoj promenljivoj, vraća se **MEMORIJSKA ADRESA** objekta (a ne sam objekt) na koji ona ukazuje!
- ▶ Kada se referencna promenljiva više ne koristi, referenca objekta se **UNIŠTAVA** ali **NE I SAM OBJEKT** (mada ovo ima za posledicu oslobađanje pridružene memorije).
- ▶ Ova procedura je slična brisanju datoteke sa hard-diska.
- ▶ Kada objekat **NEMA** svojih referenci podložan je **SKUPLJANJU OTPADAKA** (softveru koji je zadužen za skupljanje "đubreta").

# Vrednosni/referencni tipovi u memoriji



# Promenljive u Javi

- ▶ Šta su to **PROMENLJIVE** u Javi?
- ▶ **PROMENLJIVE** su oznake koje opisuju **POJEDINE LOKACIJE U MEMORIJI** kojoj je pridružen TIP **PODATAKA**.
- ▶ Promenljive su osnovna jedinica za **ČUVANJE VREDNOSTI** u Javi.
- ▶ Pre upotrebe promenljive se **MORAJU DEKLARISATI!**
- ▶ Evo nekoliko primera ispravne deklaracije elementarnih promenljivih u Javi:

```
int a, b, c;
```

```
// deklaracija tri integera a, b, and c.
```

```
int d = 3, e, f = 5;
```

```
// deklaracija i inicijalizacija celobrojne promenljive d i f.
```

```
byte z = 22;
```

```
// inicijalizacija z.
```

```
double pi = 3.14159;
```

```
// deklaracija Rudolfovog brija  $\pi$ .
```

```
char x = 'x';
```

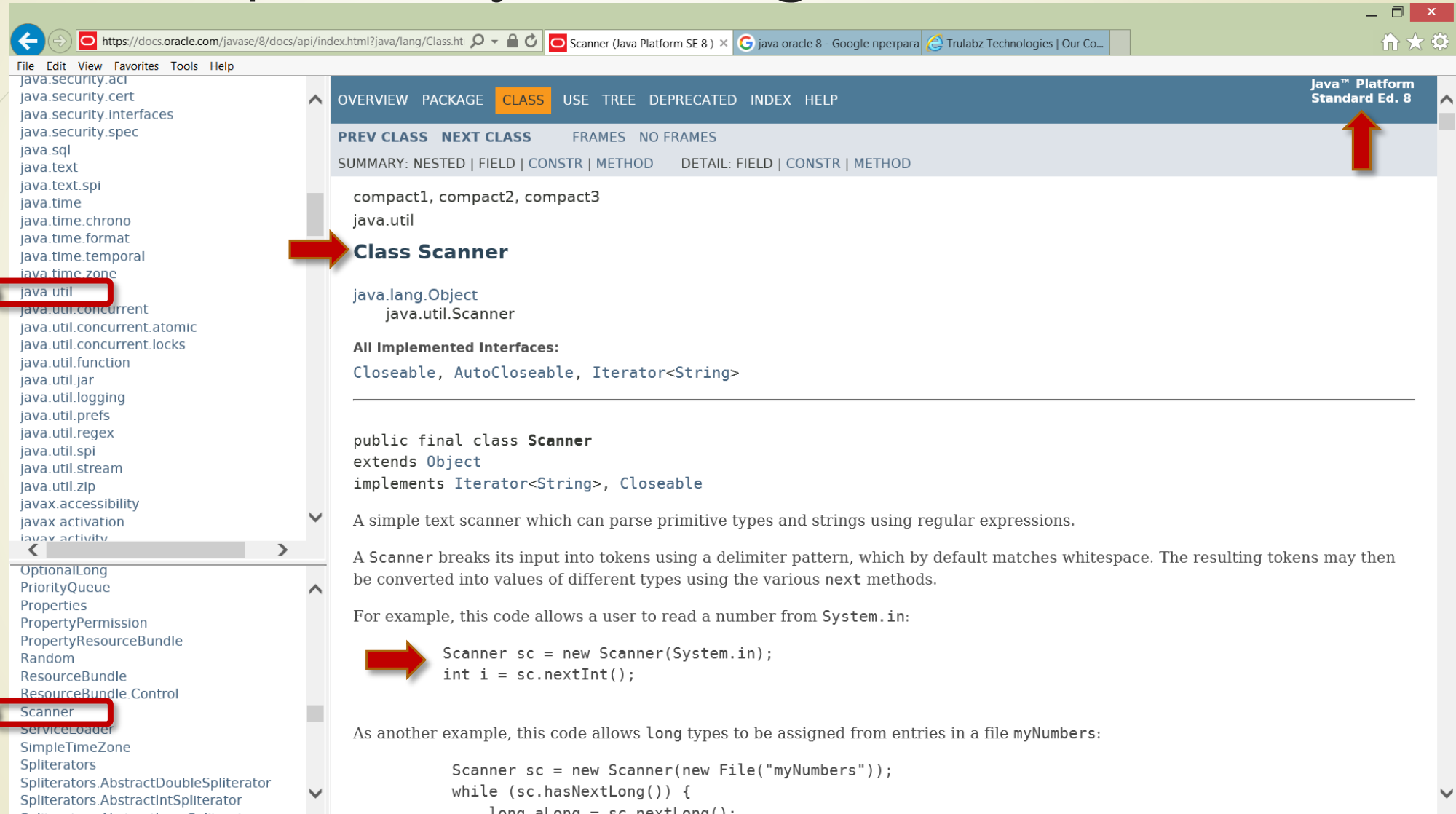
```
// promenljiva x ima vrednost 'x'.
```

# Celobrojne promenljive u Javi

```
class Example2 {  
    public static void main (String args[]) {  
        int num; // deklaracija promenljive num  
        num = 100; // pridružena vrednosti num, 100  
        System.out.println("This is num: " + num);  
        num = num * 2; // pridružena vrednosti num*2  
        System.out.print("The value of num * 2 is ");  
        System.out.println(num);  
    }  
}
```



# Oracle paketi: java.lang – Scanner



https://docs.oracle.com/javase/8/docs/api/index.html?java/lang/Class.html

Java™ Platform Standard Ed. 8

OVERVIEW PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3  
java.util

**Class Scanner**

java.lang.Object  
java.util.Scanner

**All Implemented Interfaces:**  
Closeable, AutoCloseable, Iterator<String>

public final class **Scanner**  
extends Object  
implements Iterator<String>, Closeable

A simple text scanner which can parse primitive types and strings using regular expressions.

A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.

For example, this code allows a user to read a number from System.in:

```
Scanner sc = new Scanner(System.in);  
int i = sc.nextInt();
```

As another example, this code allows long types to be assigned from entries in a file myNumbers:

```
Scanner sc = new Scanner(new File("myNumbers"));  
while (sc.hasNextLong()) {  
    long aLong = sc.nextLong();
```

# Program za prikaz zbira dva cela broja

```
import java.util.Scanner; // koristi se klasa Scanner iz java.util
public class Addition
{
    public static void main(String[] args)
    {
        // kreira se objekt input tipa Scanner da se dobije ulaz sa tastature
        Scanner input = new Scanner(System.in); // referencna promenljiva
        int number1; // prvi sabirak // vrednosna promenljiva
        int number2; // drugi sabirak // vrednosna promenljiva
        int sum; // zbir dva broja number1 i number2, vredn.promenljive
        System.out.print("Unesi prvi broj: "); // ispiši
        number1 = input.nextInt(); // učitaj prvi int. br. sa tast.
        System.out.print("Unesi drugi broj: "); // ispiši
        number2 = input.nextInt(); // učitaj drugi int. br. sa tastat.
        sum = number1 + number2; // saberi brojeve i smesti zbir u prom. sum
        System.out.printf("Zbir je %d\n", sum ); // pikaži promeljivu sum
    } // kraj metode main
} // end class Addition
// promenljiva x ima vrednost 'x'.
```

# Operatori poređenja u Javi

- ▶ Rezultat svakog poređenja je **BULOVA PROMENLJIVA** koja može uzeti vrednosti **TRUE** (istina) ili **FALSE** (laž).

| OPERATORI | ZNAČENJE OPERATORA  |
|-----------|---|
| <         | Manje od (engl. <i>less than</i> )                        |
| <=        | Manje ili jednako (engl. <i>less than or equal to</i> )   |
| ==        | Jednako (engl. <i>equal to</i> )                          |
| >=        | Veće ili jednako (engl. <i>greater than or equal to</i> ) |
| >         | Veće od (engl. <i>greater than</i> )                      |
| !=        | Nije jednako (engl. <i>not equal</i> )                    |

# Java: nizovi

- ▶ **NIZ** (engl. *array*) predstavlja **GRUPU PROMENLJIVIH ISTOG TIPA** koje se pojavljuju pod **ZAJEDNIČKIM NAZIVOM**.
- ▶ Zapamtite: Nizovi su **REFERENCNI TIPOVI** podataka!
- ▶ Koje su posledice (ili benefiti) ove činjenice?



# Java: xD nizovi

- ▶ Mogući su nizovi **SVIH TIPOVA!**
- ▶ Pristup elementima niza moguć je preko njihovog **REDNOG BROJA – INDEKSA**-a.
- ▶ Moguće je definisati sledeće tipove nizova:
  - ▶ **JEDNODIMENZIONE** (1D) i
  - ▶ **VIŠEDIMENZIONE** nizove (xD):
- ▶ Primer deklaracije 1D, 2D i 3D nizova:

```
▪ month_days[] = new int[12];           // 1D niz
▪ int twoD[][] = new int[4][5];         // 2D niz
▪ int threeD[][][] = new int[3][4][5];  // 3D niz
```

Operator **new** je neophodno primeniti prilikom formiranja **REFERENCNIH** tipova!

# Java: 2D nizovi

```
class Matrix {  
    public static void main (String args[]) {  
        double m[][] = {  
            { 0*0, 1*0, 2*0, 3*0 },  
            { 0*1, 1*1, 2*1, 3*1 },  
            { 0*2, 1*2, 2*2, 3*2 },  
            { 0*3, 1*3, 2*3, 3*3 }  
        };  
        int i, j;  
        for(i=0; i<4; i++) {  
            for(j=0; j<4; j++)  
                System.out.print(m[i][j] + " ");  
            System.out.println();  
        }  
    }  
}
```

Dekleracija i inicijalizacija  
2D niza

Štampanje elemenata  
2D niza !

# Znakovni nizovi u Javi

- ▶ Java podržava **ZNAKOVNE NIZOVE** svojom klasom **String**.
- ▶ **STRING JE OBJEKT**, što podrazumeva da su mu pridružene **METODE** za rad sa ovim tipom promenljive.
- ▶ Mogu se definisati i **NIZOVI ZNAKOVNIH NIZOVA** (setite se deklaracije metode main, koja ima kao parametar niz znakovnih nizova:

```
main(String args[])
```

- ▶ Objekt tipa string se može koristiti kao ARGUMENT metode **println()**.

```
String str = "Inicijalizacija string promenljive";
```

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

- ▶ Objekti tipa string poseduju **SPECIFIČNE OSOBINE** i **ATRIBUTE** koje olakšavaju njihovo korišćenje.
- ▶ Standardne klase i metode za rad sa stringovima biće obrađeni na **SLEDEĆEM** predavanju.

# Upravljanje programskim izvršenjem

```
class IfSample {  
    public static void main (String args[]) {  
        int x, y;  
        x = 10;  
        y = 20;  
        if(x < y) System.out.println("x je manje od y");           // da  
            x = x * 2;                                           // ne  
        if(x == y) System.out.println("x je sada jednako y");  
            x = x * 2;  
        if(x > y) System.out.println("x je sada veće od y");  
            // ovaj kod ispod neće ništa prikazati, zašto?  
        if(x == y) System.out.println("Ovo se neće nikada videti");  
    }  
}
```



# Upravljačke naredbe u Javi

```
class ForTest {  
    public static void main (String args[]) {  
        int x;  
        for(x = 0; x < 10; x = x + 1)  
            System.out.println("Vrednost x-a je: " + x);  
    }  
}
```

// deklaracija promenljive x

// petlja za prikaz

Telo naredbe **for**

Parametri naredbe **for**:

1. Inicijalizacija petlje, **x = 0**
2. Ispitivanje uslova izlaska iz petlje, **x < 10**
3. Korak **uvećanja** promenljive, **x = x + 1**

# Blok koda u Javi

```
/* Demonstracija blok koda. */
```

← Komentar za blok koda

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

```
        int x, y;
```

```
        y = 20;
```

```
        // For petlja u ovom slučaju predstavlja blok koda
```

← Komentar za liniju koda

```
        for(x = 0; x < 10; x++) {  
            System.out.println("Ovo je x: " + x);  
            System.out.println("Ovo je y: " + y);  
            y = y - 2;  
        }
```

```
    }
```

```
}
```

# Oblast važenja u Javi

- ▶ **OBLAST VAŽENJA** (ili vidljivosti) promenljivih je značajan pojam za **KAPSULIRANJE PODATAKA**.
- ▶ Promenljive se mogu definisati u okviru **BILO KOG BLOKA**.
- ▶ Zapamtie: **BLOK** definiše oblast važenja!
- ▶ **OBLAST VAŽENJA** određuje objekte koji će **BITI VIDLJIVI** drugim delovima programa.
- ▶ **OBLAST VAŽENJA** može biti definisan **KLASOM**, **METODOM** i **PAKETOM** (to smo videli na početku predavanja).
- ▶ Oblast važenja definisana **METODOM** počinje vitičastom zagradom [**{**].
- ▶ Promenljive definisane **UNUTAR OBLASTI** važenja **NISU VIDLJIVE** naredbama koje si definisane **IZVAN** te oblasti.

# Blok koda u Javi

// Računa površinu kruga.

```
class Area {
```

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

```
        double pi, r, a;                // deklaracija svih promenljivih
```

```
        r = 10.8;                        // prečnik kruga
```

```
        pi = 3.1416;                    // Rudolfov broj pi
```

```
        a = pi * r * r;                 // Računa površinu kruga
```

```
        System.out.println("Površina kruga je " + a);
```

```
    }
```

```
}
```

# Dodela char vrednosti u Javi

```
// Primer char tipa podataka
```

```
class CharDemo {
```

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

```
        char ch1, ch2;
```

```
//deklaracija promenljivih tipa char
```

```
        ch1 = 88;
```

```
//decimalna predstava broja
```

```
        ch2 = 'Y';
```

```
// ASCII kod za y
```

```
        System.out.print("ch1 i ch2: ");
```

```
        System.out.println (ch1 + " " + ch2);
```

```
    }
```

```
}
```

Jednostavno formatiranje izlaza

# Oracle doc java.lang.Math paket

The screenshot shows the Oracle Java API documentation for the `java.lang.Math` class. The left sidebar contains a navigation tree with the following packages and classes listed:

- java.applet
- java.awt
- java.awt.color
- java.awt.datatransfer
- java.awt.dnd
- java.awt.event
- java.awt.font
- java.awt.geom
- java.awt.im
- java.awt.im.spi
- java.awt.image
- java.awt.image.renderable
- java.awt.print
- java.beans
- java.beans.beancontext
- java.io
- java.lang**
- java.lang.annotation
- java.lang.instrument
- java.lang.invoke
- java.lang.management
- Compiler
- Double
- Enum
- Float
- InheritableThreadLocal
- Integer
- Long
- Math**
- Number
- Object
- Package
- Process
- ProcessBuilder
- ProcessBuilder.Redirect
- Runtime
- RuntimePermission
- SecurityManager

The main content area displays the **Method Summary** table, which is organized into three tabs: **All Methods**, **Static Methods**, and **Concrete Methods**. The table lists the following methods:

| Modifier and Type | Method and Description   |
|-------------------|--|
| static double     | <b>abs</b> (double a)<br>Returns the absolute value of a double value.   |
| static float      | <b>abs</b> (float a)<br>Returns the absolute value of a float value.   |
| static int        | <b>abs</b> (int a)<br>Returns the absolute value of an int value.  |
| static long       | <b>abs</b> (long a)<br>Returns the absolute value of a long value.   |
| static double     | <b>acos</b> (double a)<br>Returns the arc cosine of a value; the returned angle is in the range 0.0 through $\pi$ .  |
| static int        | <b>addExact</b> (int x, int y)<br>Returns the sum of its arguments, throwing an exception if the result overflows an int.                                  |
| static long       | <b>addExact</b> (long x, long y)<br>Returns the sum of its arguments, throwing an exception if the result overflows a long.                                |
| static double     | <b>asin</b> (double a)<br>Returns the arc sine of a value; the returned angle is in the range $-\pi/2$ through $\pi/2$ .                                   |
| static double     | <b>atan</b> (double a)<br>Returns the arc tangent of a value; the returned angle is in the range $-\pi/2$ through $\pi/2$ .                                |
| static double     | <b>atan2</b> (double y, double x)<br>Returns the angle $\theta$ from the conversion of rectangular coordinates (x, y) to polar coordinates (r, $\theta$ ). |
| static double     | <b>cbrt</b> (double a)   |

# Dinamička inicijalizacija u Javi

// Primer dinamičke inicijalizacije promenljivih.

```
class DynInit {  
    public static void main (String args[]) {  
        double a = 3.0, b = 4.0;  
        // c je dinamički inicijalizovana promenljiva  
        double c = Math.sqrt(a * a + b * b);  
        System.out.println("Hypotenuse is " + c);  
    }  
}
```

Class Math, sqrt - metoda !

Dinamička inicijalizacija promenljive c.