



Distance Learning System

# Core Java

Tipovi podataka / promenljive / operatori

# Agenda

---

- Ispis sadržaja na izlaz
- Tipovi podataka
- Promenljive
- Konstante
- Operatori

# Primer – površina pravougaonika

## (jcex062014 RectangleAreaExample)

```
package rectangleareaexample;
public class RectangleAreaExample {
    final static String measure = "cm";
    public static void main(String[] args) {
        int a = 2;
        int b = 3;
        int res = a * b;
        String message = "Rectangle area is " + a + measure
            + " * " + b + measure + " is " + res + measure;
        System.out.println(message);
    }
}
```

# Primer – površina pravougaonika

Naziv paketa

Konstanta (jcex062014)

Aritmetički operator

Operator dodele

Hard code

Operator za  
konkatinaciju stringa

Promenljiva

```
package rectangleareaexample;
public class RectangleAreaExample {
    final static String measure = "cm";
    public static void main(String[] args) {
        int a = 2;
        int b = 3;
        int res = a * b;
        String message = "Rectangle area is " + a + measure
            + " * " + b + measure + " is " + res + measure;
        System.out.println(message);
    }
}
```

Ispis na izlaz

# Ispis sadržaja na izlaz (jcex062014)

- U Javi se može na mnogo načina emitovati sadržaj na izlaz. Ali šta je to izlaz? I šta je to sadržaj?
- U nastavku, podrazumevaćemo konzolu kao osnovni izlaz aplikacije, pa je tako najjednostavniji način da ispišemo nešto u konzoli, sledeći:

```
public static void main(String[] args) {  
  
    //Standard console output without new line  
    System.out.print("Hello from Java");
```

- Ovaj kod prikazuje poruku na izlazu (u konzoli), ako ponovo upotrebimo istu naredbu, sledeća poruka biće ispisana u nastavku:

```
System.out.print("Hello from Java");  
System.out.print("Hello from Java");
```

- Izlaz:

```
run:  
Hello from JavaHello from JavaBUILD SUCCESSFUL (total time: 0 seconds)
```

# Ispis sadržaja na izlaz

---

- U najvećem broju slučajeva, nećemo želeći da nadovezujemo izlaz pomoću različitih funkcija za emitovanje, već ćemo radije hteti da svaka funkcija prikaže sadržaj u novom redu, za šta možemo upotrebiti naredbu **System.out.println**

```
System.out.println("Hello from Java with new line");  
System.out.println("Hello from Java with new line, again");
```

- Izlaz je:

```
run:  
Hello from Java with new line  
Hello from Java with new line, again  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Ispis sadržaja na izlaz pomoću Console objekta

- Prethodni način nije jedini pomoću koga možemo ispisati nešto na izlaz
- Možemo takođe upotrebiti i objekat klase **Console**
- Jednom dobavljen, ovaj objekat nam omogućava ispis na izlaz, ali takođe i preuzimanje sa izlaza

```
Console c = System.console();  
c.writer().println("Hello from Java via console object");
```

- U konzoli NetBeans-a program prijavljuje grešku

```
run:  
Exception in thread "main" java.lang.NullPointerException  
    at emittooutput.EmitToOutput.main(EmitToOutput.java:19)  
Java Result: 1  
BUILD SUCCESSFUL (total time: 0 seconds)
```

- U konzoli operativnog sistema, program funkcioniše bez greške

```
D:\test>java -jar EmitToOutput.jar  
Hello from Console object  
D:\test>
```

# Ispis sadržaja na izlaz pomoću Console objekta

---

- Obzirom na prethodni primer, Console objekat nam može omogućiti da proverimo da li je program startovan u konzoli:

```
Console c1 = System.console();  
if(c1!=null){  
    c1.writer().println("Console exist");  
}
```

- Pored ispisa podataka u konzolu, **Console** omogućava i preuzimanje podataka iz konzole:

```
Console c2 = System.console();  
String line = c2.readLine();  
c2.writer().println("Hello " + line);
```



# Primer, preuzinje i emitovanje sadržaja (jcex062014)

---

- Pokušajmo da napravimo jednostavan program koji će preuzeti dva broja (celobrojne vrednosti) od korisnika i izvršiti njihovo sabiranje
- Program bi trebao da ima sledeći izlaz:

```
D:\test>java -jar SimpleCalculator.jar
Enter first operand:
3
Enter second operand
5
Result is 8
D:\test>
```

# Primer, preuzimanje i emitovanje sadržaja

---

- Program je moguće rešiti na sledeći način:

```
public static void main(String[] args) {  
    Console console = System.console();  
    System.out.println("Enter first operand:");  
    String op1 = console.readLine();  
    System.out.println("Enter second operand");  
    String op2 = console.readLine();  
    System.out.print("Result is ");  
    System.out.println(  
        (Integer.parseInt(op1)+Integer.parseInt(op2))  
    );  
}
```

# Scanner

---

- Ako ćete samostalno eksperimentisati sa ulazom i izlazom u aplikaciju, uzmite u obzir da je pored komande (metode) `readLine`, moguće je takođe koristiti efikasniji sistem – **Scanner (`java.util.Scanner`)**:

```
Scanner scanner = new Scanner(System.in);  
System.out.println(scanner.nextInt() + scanner.nextInt());
```

- Ovaj sistem omogućava direktno preuzimanje vrednosti određenog tipa iz konzole

# Formatiranje izlaza

---

- Ukoliko nam je to potrebno, možemo formatirati određeni podataka prilikom prikazivanja
- Ovo je veoma čest slučaj prilikom emitovanja decimalnih brojeva
- Na primer, u prvoj liniji prikaza, prikazuje se neformatiran decimalni broj. Ovo nam u nekim slučajevima odgovara, ali često i ne. Na primer, ako hoćemo da prikažemo cenu, najčešće nećemo hteti da idemo preko dve decimale
- Druga linija formatira izlaz pomoću funkcije `format` i prikazuje željeni rezultat

```
double x = (113.0/112.0);
```

```
System.out.println("Hello from number " + x);
```

```
System.out.format("Hello from number %.2f%n", x);
```

1.0089285714285714

1.009

# Saznali smo

---

- Za slanje podataka na standardni izlaz (konzolu), u Javi koristimo klasu **System**.
- Osim za slanje na izlaz, System se takođe koristi i za druge bitne i zanimljive stvari. Na primer, za preuzimanje trenutnog vremena u milisekundama ili nanosekundama, za preuzimanje sistemskih promenljivih, za preuzimanje parametara aplikacije, aktivaciju garbage collector-a i slično
- Detalje o klasi System treba dodatno proučiti. Kao izvor, može se koristiti sledeća adresa:

**<http://docs.oracle.com/javase/7/docs/api/java/lang/System.html>**

# Promenljive

---

- Promenljive su privremeni kontejneri u koji se skladište neke vrednosti. Nakon što je vrednost uskladištena u memoriju, ona biva reprezentovana kroz promenljivu.
- Promenljive su dakle: **smisleni nazivi za neke vrednosti u memoriji**

# Deklarisanje, inicijalizovanje i instanciranje

---

- Da bi neka promenljiva postojala, mora se kreirati
- Proces kreiranja promenljive naziva se deklarisanje
- Sledećom linijom koda predstavljeno je deklarisanje jedne celobrojne promenljive:

```
int variableA;
```

- Deklaracija se sastoji iz dva dela. Prvi označava tip podatka koji će promenljiva predstavljati, a drugi predstavlja naziv promenljive.

# Deklarisanje, inicijalizovanje i instanciranje

- Prilikom imenovanja promenljivih postoji nekoliko pravila kojih se moramo držati:
  - Promenljive ne smeju početi brojem, ključnom rečju jezika ili specijalnim karakterom
  - Osetljive su na velika i mala slova (promenljive name i Name su različite)
  - Promenljiva mora biti jedna reč (myVariable bi bilo ispravno, ali ne i my Variable)

```
int void;  
int my Variable;  
int @myVariable;
```

Neispravno

Ispravno

```
int myVoid;  
int myVariable;  
int myVariable;
```



# Konvencije za imenovanje promenljivih

---

- Iako pri samom imenovanju promenljivih ne moramo poštovati nikakva jezička pravila, poželjno je da se koristi jedna od nekoliko notacija za pisanje. Jedna od najpopularnijih je Camel Case notacija, koja podrazumeva veliko slovo na početku svake reči u promenljivoj, kao na primer:
  - **MyVariable** ili **myFirstVariable**
- Takođe, preporučuje se intuitivno imenovanje promenljivih. Ovu preporuku treba ozbiljno shvatiti čak i prilikom konstrukcije najmanjih programa jer se i ovakvi programi veoma brzo prošire, a pamćenje informacija o promenljivima – se lako otme kontroli.

# Inicijalizacija promenljivih

---

- Inicijalizacija promenljive je dodela vrednosti.
- Promenljive se mogu prvo deklarirati a zatim inicijalizovati, ili inicijalizovati prilikom deklaracije

```
int variableA; //declaration  
variableA=10; //initialization  
int variableB = 10; // declaration and initialization
```

# Inicijalizacija (instanciranje) objekata

---

- Instanciranje je proces kreiranja objekta i rezervisanja memorije koju će koristiti jedan objekat

```
MyClass myClass;  
myClass = new MyClass();  
MyClass myClass1 = new MyClass();
```

# Tipovi podataka

---

- Tipovi podataka su šabloni koji se koriste prilikom unosa nekih vrednosti u program
- Gotovo svaki programski jezik poznaje pojam **tipova podataka** pa tako i Java
- U većini jezika, tipovi podataka se dele na **proste** i **složene**
- Sami jezici se takođe mogu podeliti po tipovima na **strogo tipizirane** i **slabo tipizirane**

# Tipovi podataka u Javi

---

- Java je **statički tipiziran** jezik
  - Prostor za promenljive se alocira prilikom startovanja programa
- Java je **strogo tipiziran** jezik
  - Promenljive koje se deklarišu pod određenim tipom, ne mogu skladištiti vrednosti drugog tipa

# Primitivni tipovi u Javi

<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

---

- **byte** – 1 bajt –  $2^7$
- **short** – 2 bajta –  $2^{15}$
- **int** – 4 bajta –  $2^{31}$
- **long** – 8 bajtova –  $2^{63}$
- **float** – 4 bajta
- **double** – 8 bajtova
- **boolean** – 1 bajt
- **char** – 2 bajta –  $2^{15}$

# Primitivni tipovi u Javi

<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

- **byte** – 1 bajt
- **short** – 2 bajta
- **int** – 4 bajta
- **long** – 8 bajtova
- **float** – 4 bajta
- **double** – 8 bajtova
- **boolean** – 1 bajt
- **char** – 2 bajta



# Upotreba tipova podataka (jcex062014 PrimitiveTypes)

- U Java programu, obavezno je deklarirati tip za neku vrednost pre njene upotrebe
- Deklaracija podrazumeva naziv tipa, a zatim i naziv promenljive
- Nakon naziva tipa i promenljive, može a ne mora da postoji dodela vrednosti

```
byte crew_exp = 98;
short engine_power = 700;
float rate_of_fire = 8.96f;
boolean showmark;
showmark = true;
char[] tank_name = {'V','K',' ','3','6','.', '0','1',' ','(',')','H','('};
```



# Upotreba tipova podataka (jcex062014 PrimitiveTypes1)

- Prilikom dodele vrednosti, kada su u pitanju numerički tipovi, moguće je koristiti različite brojevne notacije

```
int x = 10;  
int y = 010;  
int z = 0x10;  
int i = 0b10;  
System.out.println(x);  
System.out.println(y);  
System.out.println(z);  
System.out.println(i);
```



```
run:  
10  
8  
16  
2  
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Ovako napisan program, daje zanimljive (i možda neočekivane) rezultate. Zašto?



# Upotreba tipova podataka (jcex062014 PrimitiveTypes1)

```
int x = 10;  
int y = 010;  
int z = 0x10;  
int i = 0b10;  
System.out.println(x);  
System.out.println(y);  
System.out.println(z);  
System.out.println(i);
```



```
run:  
10  
8  
16  
2  
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Zato što su prilikom dodele vrednosti promenljivima, upotrebljene različite brojevne notacije, pa je tako:
- $10 = 10$ , jer se koristi dekadna notacija (10 je 10)
- $010 = 8$  jer se koristi oktalna notacija (10 je oktalno 8)
- $0x10 = 16$  jer se koristi heksadecimalna notacija (10 je hex 16)
- $0b10 = 2$  jer se koristi binarna notacija (10 je binarno 2)

# Podrazumevane vrednosti prostih tipova

---

- byte **0**
- short **0**
- int **0**
- long **0L**
- float **0.0f**
- double **0.0d**
- char **'\u0000'**
- boolean **false**

# Wrapper klase primitivnih tipova

- Svaki od osam primitivnih tipova poseduje odgovarajuću klasu koja omogućava određene dodatne mogućnosti nad primitivnim tipovima, koje u protivnom ne bi bile moguće
- Kako i samo ime kaže, o ovim klasama se može razmišljati kao o određenom „omotaču“ koji objedinjuje primitivne tipove i paket dodatnih funkcionalnosti nad njima

Primitivni tip	Klasa
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

# Konverzije prostih tipova

---

- **Implicitna konverzija** je konverzija koja se obavlja automatski od strane JVM-e, i događa se u slučajevima kada pokušavamo da konvertujemo podatak manje veličine u veći. U ovom slučaju dolazi do proširivanja manjeg podatka i konverzija se obavlja bez bilo kakvog gubitka informacija.

```
int x = 10;           // 4 bytes
double y = x;         // 8 bytes
System.out.println(y);
```

# Konverzije prostih tipova

- Ukoliko bismo hteli da izvršimo konverziju većeg tipa u manji, ne bismo mogli da se oslonimo da će JVM-a obaviti posao automatski za nas
- Sam čin konvertovanja podatka koji okupira više memorije u tip koji zauzima manje memorije je problematičan, jer postoji mogućnost gubitka informacija (zbog toga prevodilac očekuje od nas da tip konvertujemo **eksplicitno**)

```
4 | | double x = 10; // 8 bytes  
5 | | int y = x; // 4 bytes  
6 | | System.out.println(y);
```

# Konverzije prostih tipova

---

- **Eksplicitna konverzija** tipova podrazumeva da „kažemo“ prevodiocu da želimo da prevedemo neku promenljivu iz jednog tipa u drugi
- Ovo je moguće uraditi na više načina (u zavisnosti od tipa), ali se za proste tipove obično koristi standardna notacija za konverziju:

```
double x = 10.5;  
int y = (int) x;  
System.out.println(y);
```

- Navođenjem tipa u koji želimo da izvršimo konverziju u zagradama ispred naziva promenljive vršimo eksplicitnu konverziju

# String

---

- U programskom jeziku Java za rukovanje tekstualnim vrednostima koristi se tip String.
- Mnogo je zgodnije rukovati kompletnim rečima, ili rečenicama nego karakterima, pa iz tog razloga ovaj tip podatka postoji u gotovo svakom programskom jeziku.
- Za tip String kažemo i da je specijalni tip, pošto ima osobine prostog tipa, iako je zapravo objektni tip
- Ovo je jedini objektni tip koji se može kreirati kao i svaki drugi prosti tip.
- Ipak ovaj tip, zbog svoje objektne osobenosti poseduje pregršt metoda i svojstava koji olakšavaju rad sa vrednostima ovog tipa.

```
String message = "Hello!!!";  
System.out.println(message);
```



# Složeni tipovi

---

- Složeni tipovi obično podrazumevaju više informacija na jednom mestu
  - **Nizovi**
  - **Klase**
  - **Interfejsi**
  - **Objekti**
- Kompleksni tipovi su najčešće i referencni tipovi

# Nepostojeći tip – null

---

- U javi postoji oznaka null, koja prestavlja nedostatak vrednosti. Ona predstavlja referncu koja ne pokazuje ni na koji objekat
- objekat ima vrednost null (odnosno nema vrednost), to znači da kreirana objektna promenljiva ne pokazuje ni na jednu memorijsku lokaciju

```
String message = null;  
System.out.println(message);
```



```
run:  
null  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Vežba 1 (jcex062014 ex1)

---

- U jednom programu potrebno je čuvati vrednost godina i visinu plate jednog radnika. Deklarisati promenljive koje mogu da posluže u svrhu čuvanja ovih vrednosti

# Operatori

---

- Da bismo stvorili jedan koliko-toliko funkcionalan program, u bilo kom jeziku, potrebni su nam, pored promenljivih i operatori.
- I ovde postoji generalna klasifikacija na: **operatore dodeljivanja, aritmetičke operatore, operatore poređenja i logičke operatore**. Postoji još nekoliko vrsta operatora (String operatori, Type operatori, inkrement-dekrement operatori), ali se oni generalno, baziraju na pomenuta četiri osnovna operatora.

# Operatori dodeljivanja

---

- U Javi (a i u ostalim jezicima), zapravo, postoji samo jedan operator dodeljivanja:

=

i sa njim smo se već upoznali u prethodnim lekcijama.

- Operator = znači da će sve što se nalazi sa leve strane, dobiti vrednost onoga što se nalazi sa desne strane

```
int x = 10;  
String message = "Hello from Java!!!";
```

# Aritmetički operatori

---

- Sabiranje :  $+$
- Oduzimanje :  $-$
- Množenje :  $*$
- Deljenje :  $/$
- Modulo/moduo (celobrojni ostatak od deljenja) :  $\%$

Četiri operatora standardnih aritmetičkih operacija nije potrebno bliže objašnjavati. Kao i u matematici, to su operatori za sabiranje, oduzimanje, množenje i deljenje.

# Aritmetički operatori

```
int x = 10;  
int y = 5;  
System.out.println(x+y);  
System.out.println(x-y);  
System.out.println(x*y);  
System.out.println(x/y);
```


*Kod sa leve strane  
daje sledeći rezultat*



```
run:  
15  
5  
50  
2  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Operator % predstavlja celobrojni ostatak deljenja dva broja

```
int modulo = 10 % 3;  
System.out.println(modulo);
```



```
run:  
1  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Na isti način možemo isprobati funkcionisanje ovog operatora i u drugim situacijama.

10%4 daje kao rezultat vrednost 2.

10%6 daje vrednost 4.

# Aritmetički operatori

---

- Prilikom izvršavanja matematičkih izraza, veoma je važno znati za postojanje prioriteta operatora
- Množenje i deljenje uvek imaju prioritet u odnosu na sabiranje i oduzimanje

```
int a = 2, b = 3, c = 4;  
int res = a * b + c - a;  
System.out.println(res);  
res = a * (b + c - a);  
System.out.println(res);
```

- *Pokušajte na osnovu ovog saznanja da pogodite kako će i zbog čega izgledati izlaz prethodnog programa*



# Konkatinacija stringova

---

- Aritmetički operator `+`, pored toga što se koristi za sabiranje brojčanih vrednosti u jeziku Java ima još jednu funkciju. Ova funkcija se ogleda u mogućnosti sabiranja tekstualnih vrednosti, što se naziva nadovezivanje ili konkatenacija stringova. Već smo rekli da tip podatka `String` ima specijalan tretman u Javi, pošto je zapravo objektni, ali se njim može rukovati kao da je primitivni. Kada se ovo kaže, misli se i na primenu aritmetičkog operatora `+` nad stringovima. Pogledajmo primer:

```
String s1 = "Hello ";  
String s2 = "from string";  
System.out.println(s1 + s2);
```

# Unarni operatori

---

- Unarni operator za svoje funkcionisanje zahtevaju samo jedan operand. Pregled unarnih operatora dat je u sledećoj tabeli:

Operator	Značenje
+	unarni plus operator (označava pozitivnu vrednost)
-	unarni minus operator (označava negativnu vrednost)
++	operator uvećanja; vrši uvećanje za jedan
--	operator umanjenja; vrši umanjenje za jedan
!	invertuje vrednost boolean promenljive

- Operatori **+** i **-** pretvaraju operand u pozitivan, odnosno negativan broj

# Inkrement / dekrement

- Inkrement / dekrement operatori omogućavaju povećavanje, odnosno, smanjivanje promenljive za jedan
- Ovaj operator može se pojaviti u jednom od dva oblika

**1** `int x = 10;`  
`x++; // dodavanje broja 1 na x`  
`x--; // oduzimanje broja 1 od x`

**2** `int x = 10;`  
`++x; // dodavanje broja 1 na x`  
`--x; // oduzimanje broja 1 od x`

- Oba slučaja imaju isti epilog kada je u pitanju promenljiva x, koja će biti povećana i smanjena za jedan (što znači da će biti prvo 11, a zatim opet 10)
- Za razliku od samog x, ova dva koda mogu imati veoma različit epilog po „svet“ oko ove promenljive ↓

# Inkrement / dekrement



- U sledeća dva primera, konačni epilog je uvek isti po x, ali je drugačiji po promenljivu y:

```
int x = 10;  
int y = ++x;  
System.out.println(x);  
System.out.println(y);
```



```
run:  
11  
11  
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
int x = 10;  
int y = x++;  
System.out.println(x);  
System.out.println(y);
```



```
run:  
11  
10  
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Ovo je očekivana pojava. U prvom slučaju, inkrement je izvršen pre dodele vrednosti, a u drugom nakon dodele nje

# Inkrement / dekrement

- Prethodno opisano ponašanje inkrement-a i dekrementa, može imati veoma ozbiljne posledice po aplikaciju, ukoliko se na njih ne obrati pažnja. Na primer:

```
int a = 2, b = 3;  
add(a, b++);  
add(a, ++b);
```

Rezultat će prvi put  
biti 5 a drugi put 7

```
int a = 2, b = 3;  
int res = a * (b++) + b;  
int res1 = a * (++b) + b;  
System.out.println(res);  
System.out.println(res1);
```

Dva rezultata (res i res1) će se dramatično razlikovati nakon izvršenja koda  
*Pokušajte da pogodite koji će biti rezultat u oba slučaja, i zbog čega*

# Negacija

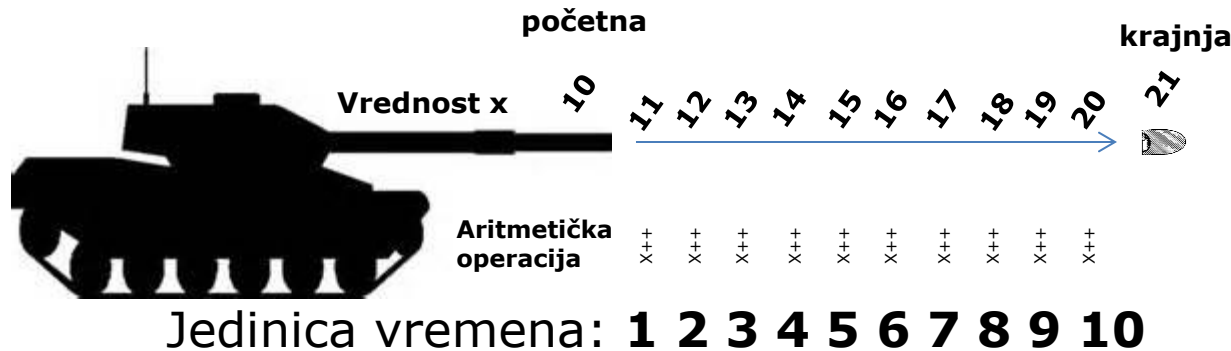
---

- Operator negacije se predstavlja oznakom **!**
- Operator negacije negira izjavu koja sledi:
  - !danas je lep dan (znači da danas **nije** lep dan)
  - !ja ne volim Javu (znači **ja volim** Javu)
  - !!!!!!!!!!!!!!!!!!!!!!!World of Warcraft je bolji od Eve Online
- Operator negacije se najčešće koristi u saradnji sa **boolean** tipovima
- Pogledajmo kako bi izgledala implementacija prethodnog primera:

```
boolean isbetter = !!!!!!!!!!!!!!!!!!!!!!!true;  
System.out.println(isbetter);
```

# Čemu nam mogu poslužiti operatori?

- Recimo da u igri treba da pomeramo projektil
- Problem je u tome što program ne zna ni šta je tenk, ni šta je projektil. Jedino što zna, to je da postoje dva crteža (ono što mi vidimo kao tenk i projektil). Ti crteži se nalaze na nekim pozicijama ekrana. Da bi se stekao utisak kretanja objekta, moramo u jedinici vremena izvršiti izmenu pozicije slike koja se kreće (projektil u ovom slučaju)



# Operatori poređenja

---

- Ovi operatori služe za poređenje vrednosti operanada i kao rezultat daju isključivo vrednost tipa boolean

Operator	Značenje
==	jednako je
!=	nije jednako
<	manje od
>	veće od
>=	veće ili jednako od
<=	manje ili jednako od



# Operator == (jcex062014 CompareOperators)

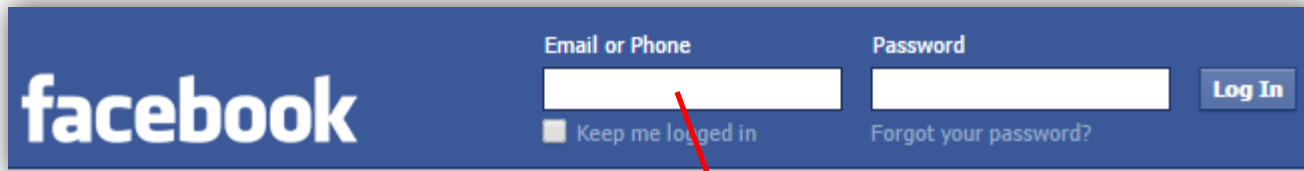
- Ovaj operator proverava da li je leva strana jednaka desnoj strani i vraća rezultat tipa boolean (true ili false)

```
int x = 10;
int y = 20;
boolean res = x == y;
System.out.println(res);
```

- Prilikom poređenja jednakosti treba paziti na uobičajenu grešku, upotrebu operatora poređenja umesto operatora jednakosti

```
boolean x1 = false;
if(x1=true) {
    System.out.println("This shouldn't be shown");
}
```

# Čemu nam može poslužiti operator ==

A screenshot of the Facebook login interface. It features the Facebook logo on the left, followed by two input fields labeled 'Email or Phone' and 'Password'. Below the 'Email or Phone' field is a checkbox labeled 'Keep me logged in'. To the right of the 'Password' field is a link that says 'Forgot your password?'. A 'Log In' button is positioned to the right of the 'Password' field. A red arrow points from the 'Email or Phone' input field down towards the code block below.

- Prilikom provere korisničkih podataka, česte se koristi upravo operator poređenja ==

```
if (username==dbusername) {  
    System.out.println("Successful login");  
}
```

Ovaj primer je banalizovan i provera korisnika na facebooku nije uistinu ovakva

# Operatori <, <=, > i >=

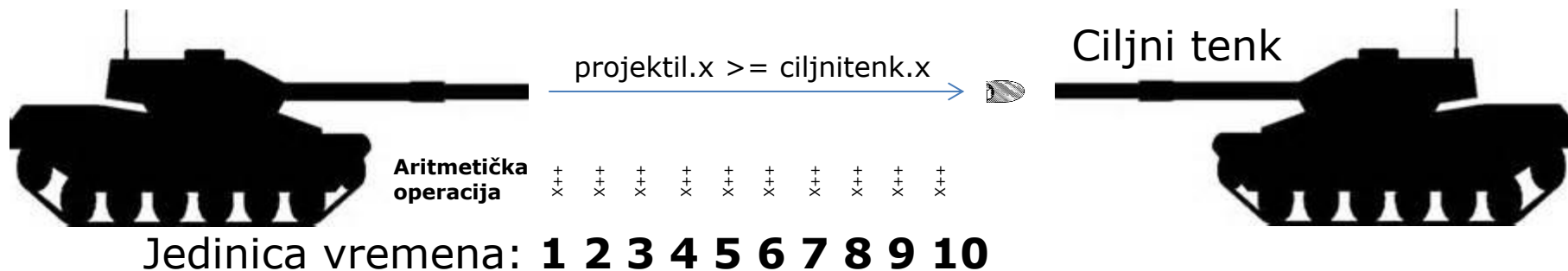
- Ove operatore koristimo da proverimo da li je neka vrednost veća od druge, ili veća ili jednaka sa drugom

```
//operator <
res = x < y;
System.out.print("x lower than y : ");
System.out.println(res);
//operator >
res = x > y;
System.out.print("x greater than y : ");
System.out.println(res);
```

```
//operator <=
res = x <= y;
System.out.print("x lower or equals y : ");
System.out.println(res);
//operator >=
res = x >= y;
System.out.print("x greater or equals y : ");
System.out.println(res);
```

# Čemu nam mogu poslužiti operatori $<$ i $>$

- Podsetimo se tenka od malopre
- Njegov projektil leti kad drugom tenku, ali kako ćemo znati da li ga je pogodio ili ne?
- Tako što ćemo u svakoj vremenskoj jedinici proveravati da li je  $x$  pozicija projektila veća ili jednaka  $x$  poziciji ciljnog tenka
- Ako je pozicija  $x$  projektila veća od pozicije  $x$  ciljnog tenka, smatracemo da je ciljni tenk pogodjen



# Čemu nam mogu poslužiti operatori < i >

## (jcex062014 CompareOperators1)

---

- Pokušajmo da kreiramo program koji će biti implementacija primera sa tenkovima sa prethodnog slajda.
- Možemo realizovati program tako što ćemo definisati promenljive x za ciljni tenk i projektil, a zatim povećavati x poziciju projektila za jedan i ispisivati da li je pozicija projektila veća od pozicije tenka.

```
int target_tank_x = 4;
int projectile_x = 0;
projectile_x++;
System.out.println("Collission: " + (projectile_x>target_tank_x));
projectile_x++;
System.out.println("Collission: " + (projectile_x>target_tank_x));
...
```

# Logički operatori

---

- Logički operatori koriste se za spajanje više izraza kreiranih upotrebom operatora poređenja
- Postoje tri uslovna operatora u Javi:

Operator	Značenje
&&	AND
	OR
?:	ternarni operator (skraćeno if-then-else)

- Prva dva operatora prikazana u tabeli su logički "I" i logički "ILI" operator.

# Logički operatori

## Prikaz korišćenja AND operatora

	Upotreba logičkog AND operatora &&	
true	true	true
true	false	false
false	false	false

## Prikaz korišćenja OR operatora

	Upotreba logičkog OR operatora	
true	true	true
true	false	true
false	false	false

## Implementacija AND operatora

```
int x = 10;
int y = 20;
boolean res = x<20&&x>10;
System.out.println(res);
res = x<20||x>10;
System.out.println(res);
```

# Bočni efekti kod logičkih operatora

---

- Kod logičkih operatora prepoznamo jednu važnu osobenost – **bočne efekte**
- Kod operatora ILI (`||`), ukoliko je prvi uslov ispunjen, drugi se i ne proverava
- Kod operatora I (`&&`), ako prvi uslov nije ispunjen, drugi se neće proveravati
- Ova pojava može dovesti do neželjenih efekata po tok programa, ukoliko na nju ne obratimo pažnju



# Bočni efekti kod logičkih operatora

## (jcex062014 LogicalOperatorsSideEffects)

---

- Na primer, u sledećem programu bi mogli očekivati da će na izlazu biti napisano true i 21, jer x jeste 10 i y jeste 20.

```
int x = 10;  
int y = 20;  
boolean res = (x==10||++y==20);  
System.out.println(res);  
System.out.println(y);
```

Nikada se ne izvrši



- Umesto očekivanog rezultata, na izlazu će biti ispisano: true i 20, jer program nikada nije stigao do inkrementa promenljive y

# Bočni efekti kod logičkih operatora

## (jcex062014 LogicalOperatorsSideEffects)

---

- U sledećem programu bi mogli očekivati da će na izlazu biti napisano false i 21, jer x nije 10

Nikada se ne izvrši

```
res = (x==11&&++y==20);  
System.out.println(res);  
System.out.println(y);
```

- Umesto očekivanog rezultata, biće prikazano false i 20, jer drugi deo uslova nikada nije izvršen

# Bočni efekti kod logičkih operatora

## (jcex062014 LogicalOperatorsSideEffects)

---

- Izrazi kreirani pomoću kombinacije logičkih operatora i operatora poređenja mogu biti veliki i komplikovani pa je to dodatni razlog da se obrati pažnja na bočne efekte
- Najbolje je svaku intervenciju na promenljivima za svaki slučaj uraditi izvan izraza

```
y++;  
res = (  
  (x>9&&x!=25&&y>19&&x!=10) ||  
  y!=20&&y==10 || (x==1&&x==2&&x==3)  
  || true  
);  
System.out.println(res);
```

- Pokušajte da pogodite koji će biti izlaz ovog koda

# Čemu nam mogu poslužiti logički operatori

- U prethodnom primeru, proverili smo da li je pozicija x projektila veća od pozicije x ciljnog tenka. To znači da bi se sledeća situacija mogla tretirati kao pogodak (što zapravo nije tačno):

X projektila je veći od x tenka  
Pa se to smatra pogotkom



- Da bi uistinu proverili da li je projektil pogodio tenk, moramo proveriti x i y koordinate projektila. Zanima nas da li je x projektila veći od x pozicije tenka, i da li je y projektila veći od y pozicije tenka

X projektila je veći od x tenka  
i y projektila je veći od y tenka  
Pa se to takođe smatra pogotkom



# Čemu nam mogu poslužiti logički operatori

- Da bi ispravno detektovali da li je projektil pogodio tenk, treba proverimo da li je:
  - x projektila veći od x tenka **I**
  - x projektila manji od x tenka + širina tenka **I**
  - y projektila veći od y tenka **I**
  - y projektila manji od y tenka + visina tenka



Samo ako se projektil nalazi u okviru tenka on se smatra pogodenim

```
int projectile_x = 10, projectile_y = 15;
int tank_x = 12, tank_y = 12, tank_width = 10, tank_height = 5;
boolean hit = (
    projectile_x > tank_x && projectile_x < (tank_x + tank_width) &&
    projectile_y > tank_y && projectile_y < (tank_y) + tank_height
);
System.out.println("Tank is hit: " + hit);
```

# Operatori nad bitovima

---

- Operatori nad bitovima se mogu koristiti samo na prostim tipovima
- Najčešće, ovi operatori se koriste za različite tipove „maskiranja“, a ređe za izmenu strukture jednog bajta
- Operatori nad bitovima su:
  - Shift -  $\ll$  i  $\gg$
  - Bit OR -  $|$
  - Bit AND -  $\&$
  - Bit XOR -  $\wedge$
  - Inverzija -  $\sim$

# Upotreba bit operatora (<<,>>)

- Bit shift operator pomera bitove ulevo ili udesno

```
int x = 2;  
x = x << 2;  
System.out.println(x);
```

- Rezultat prethodnog primera će biti 8. Da bi razumeli zašto, pogledajmo bitove u ovom programu:
  - Prvobitno stanje x: **2** je binarno **0010**
  - Nakon pomeranja bitova u levo za dva mesta (**x<<2**) broj je binarno dobio sledeću strukturu: **1000**
  - Nakon prikaza broja u decimalnoj notaciji, dobijamo broj **8**
- Po istom principu funkcioniše i pomeranje bitova u desnu stranu

# Upotreba bit operatora (|,&^)

- AND, OR i XOR su obični logički operatori koji se ne odnose na kompletne vrednosti, već na bitove tih vrednosti






```
int x = 1;  0001
int y = 2;  0010
int z = x & y;  0000
System.out.println(z);
z = x | y;  0011
System.out.println(z);
z = x ^ y;  0011
System.out.println(z);
```

Diagram illustrating the use of bit operators (AND, OR, XOR) on binary values. The code shows the assignment of variables x and y, followed by the calculation of z using these operators. The binary representations of the values are shown next to the variables, and the results of the operations are shown next to the variable z.

Variable	Value	Binary Representation
x	1	0001
y	2	0010
z (x & y)	0	0000
z (x   y)	3	0011
z (x ^ y)	3	0011



# Čemu nam mogu poslužiti bit operatori

- Bit operatori se veoma često koriste za proveru statusa neke komponente sistema, ili za maskiranje. Recimo da želimo ciljati tenk iz prethodnog primera da podelimo na tri celine: top, telo i kupola:
- Možemo napraviti tri bit maske:
- Top može biti 1, kupola 2 i telo 4
- Binarna reprezentacija ovih maski izgleda ovako:



Top 0001  
Kupola 0010  
Telo 0100

- Ako bi hteli da projektil reaguje samo na telo i kupolu (ali ne i na top), mogli bi reći da je maska kolizije: 0110 (0001 je top i 0010 je kupola)
- Nakon toga, lako bi mogli uporediti masku kolizije i sa statusom projektila: 0110 & status\_projektila

# Čemu nam mogu poslužiti bit operatori

(jcex062014 BitMasks)

- Implementacija prethodnog primera:

0001 0010 0100

```
int cannon = 1, turret = 2, body = 4;
int collision_mask = turret | body;
int bullet_state = 0;

System.out.println(bullet_state & collision_mask); → 0000 & 0110 = 0000
bullet_state |= cannon; → 0000 | 0001 = 0001
System.out.println(bullet_state & collision_mask); → 0001 & 0110 = 0000
bullet_state |= turret; → 0001 | 0010 = 0011
System.out.println(bullet_state & collision_mask); → 0011 & 0110 = 0010
bullet_state &= ~turret; → 0011 & (1101) = 0001
System.out.println(bullet_state & collision_mask); → 0001 & 0110 = 0000
```

# Čemu nam mogu poslužiti bit operatori

- Bit operatori se takođe koriste za „data parity check“
- Kod parity check-a se za svaka dva podatka čuva treći podatak (parity), pomoću koga se uvek može povratiti jedan od podataka u slučaju gubitka onog drugog
- Parity check se često koristi za obezbeđivanje integriteta podataka (na primer RAID)

```
int data_1 = 0b00001010;  
int data_2 = 0b10010101;  
int parity = data_1^data_2;  
System.out.println(data_2^parity);  
System.out.println(data_1^parity);
```

# Zadatak 1

---

- Potrebno je kreirati program za prikaz nivoa pristupa korisnika
- Prilikom startovanja aplikacije, korisnik mora da unese svoj nivo pristupa (1,2 ili 4)
- Nakon unosa program prikazuje da li kojim nivoima korisnik ima pristup a kojima ne ako su nivoi korisnika (regular=1, admin=2, superadmin = 4)
- Program može biti rešen sa hard kodiranom vrednošću ili pravim unosom korisnika
- Izlaz:

```
User level is 2  
Regular user level: false  
Admin user level: true  
Superadmin user level: false
```

# Zadatak 2

---

- Napraviti program koji će na izlazu prikazati da li je vrednost promenljive koju je korisnik uneo parna. Takođe, treba da bude prikazana uneta vrednost
- Program može biti rešen sa hard kodiranom vrednošću ili pravim unosom korisnika
- Izlaz:

```
Entered variable is 1  
Variable is even false
```