Prosti tipovi podataka

Operatori

- **Logički operatori**

!	logička negacija		
&&	logička konjunkcija	**&**	konjunkcija nad bitovima
||	logička disjunkcija	**|**	disjunkcija nad bitovima
		^	ekskluzivna disjunkcija nad bitovima

- Operator `!` je unaran, dok su svi ostali binarni
- Operatori s leve strane primenljivi su isključivo na operandima tipa `boolean`
- Operatori s desne strane su istovremeno i logički operatori (primenljivi na tip `boolean`), i operatori koji rade na bitovima (primenljivi na celobrojnim tipovima)

Logički operator !

- Neka je `a` operand (promenljiva ili izraz) tipa `boolean`
- Istinitosna tablica za operator `!`:

`!a`

a	Rezultat
true	false
false	true

- Operator `!` daje suprotan rezultat u odnosu na operand

Logički operator ! : primer

```
class TestNOT {
  public static void main(String[] args) {
    boolean a = true;
    boolean b = false;
    System.out.println(!a);
    System.out.println(!b);
  }
}
```

- Izlaz:

```
false
true
```

Logički operatori

- Neka su `a` i `b` operandi (promenljive ili izrazi) tipa `boolean`
- Istinitosne tablice za operatore `&&` i `&`, odnosno `||` i `|`

`a && b, a & b`

a	b	Rezultat
true	true	true
true	false	false
false	true	false
false	false	false

`a || b, a | b`

a	b	Rezultat
true	true	true
true	false	true
false	true	true
false	false	false

Logički operatori

- Za tip `boolean` operatori `&&` i `&`, odnosno `||` i `|` daju iste rezultate

- Postoje važne **razlike**:
  - `&&` i `||` se primenjuju samo nad tipom `boolean`
  - `&` i `|` se mogu primeniti nad tipom `boolean`, ali i nad svim celobrojnim tipovima, i tada rade po pojedinačnim bitovima, na primer:

```
System.out.println(7 & 9); // 1
System.out.println(7 | 9); // 15
```

  - `&&` i `||` se evaluiraju **lenjim izračunavanjem** (*short-circuiting*), što znači da ako se vrednost izraza može zaključiti na osnovu prvog operanda, drugi operand se neće ni izračunavati; kod `&` i `|` se uvek izračunavaju svi operandi (**vredno izračunavanje**)

Logički operatori && i &: primer

```
class TestAND {
  public static void main(String[] args) {
    int i = 0;
    int j = 10;
    boolean test;
    test = (i > 10) && (j++ > 9);
    System.out.println(i);
    System.out.println(j);
    System.out.println(test);
    test = (i > 10) & (j++ > 9);
    System.out.println(i);
    System.out.println(j);
    System.out.println(test);
  }
}
```

- Izlaz:

```
0
10
false
0
11
false
```

Logički operatori || i |: primer

```
class TestOR {
  public static void main(String[] args) {
    int i = 0;
    int j = 10;
    boolean test;
    test = (i < 10) || (j++ > 9);
    System.out.println(i);
    System.out.println(j);
    System.out.println(test);
    test = (i < 10) | (j++ > 9);
    System.out.println(i);
    System.out.println(j);
    System.out.println(test);
  }
}
```

- Izlaz:

```
0
10
true
0
11
true
```

Logički operator ^

- Istinitosna tablica za operator ^

a ^ b

a	b	Rezultat
true	true	false
true	false	true
false	true	true
false	false	false

- Operator ekskluzivne disjunkcije ^ daje rezultat `true` ako je tačno jedan operand `true`, a drugi `false`
- Pošto je neophodno izračunati vrednosti oba operanda da bi se došlo do rezultata, lenjo izračunavanje nema smisla (i zato ne postoji operator ^^)

Logički operator ^: primer

```
class TestXOR {
  public static void main(String[] args) {
    boolean a = true;
    boolean b = true;
    System.out.println(a ^ b);
    a = true; b = false;
    System.out.println(a ^ b);
    a = false; b = true;
    System.out.println(a ^ b);
    a = false; b = false;
    System.out.println(a ^ b);
  }
}
```

- Izlaz:

```
false
true
true
false
```

Operatori

- **Operatori dodele**

`=`	dodela
`+=`	operator dodele sa prethodnom primenom operatora `+`
`-=`	operator dodele sa prethodnom primenom operatora `-`
`*=`	operator dodele sa prethodnom primenom operatora `*`
`/=`	operator dodele sa prethodnom primenom operatora `/`
`%=`	operator dodele sa prethodnom primenom operatora `%`
`<<=`	operator dodele sa prethodnom primenom operatora `<<`
`>>=`	operator dodele sa prethodnom primenom operatora `>>`
`>>>=`	operator dodele sa prethodnom primenom operatora `>>>`
`&=`	operator dodele sa prethodnom primenom operatora `&`
`|=`	operator dodele sa prethodnom primenom operatora `|`
`^=`	operator dodele sa prethodnom primenom operatora `^`

Operatori dodele

- Osnovni operator = koristi se u obliku:
  `promenljiva = vrednost`
- Promenljiva sa leve strane znaka = mora biti već deklarisana (ili se operator koristi prilikom deklaracije)
- Vrednost sa desne strane može biti neki **literal** (npr. `2`), **promenljiva**, **poziv metoda**, odnosno u opštem slučaju **izraz** koji kombinuje literale, promenljive i pozive metoda
- Način izvršavanja: prvo se izračuna vrednost izraza sa desne strane =, zatim se ta vrednost dodeli promenljivoj sa leve strane
- Takođe, cela konstrukcija `promenljiva = vrednost` predstavlja izraz koji ima vrednost jednaku dodeljenoj vrednosti

  Primer:
  ```
  int i;
  int j = (i = 22) + 8;
  ```
  - Ovu mogućnost u principu treba izbegavati, jer može dovesti do grešaka koje se lako prave a teško uočavaju

Operatori dodele

- Operatori oblika `op=` , gde je `op` neki od navedenih operatora (+, –, ...) koriste se u obliku:
  `promenljiva op= vrednost`

- Izvršavaju se isto kao
  `promenljiva = promenljiva op vrednost`

- Primer:

```
int i = 2;
i += 2; // i = i + 2;
i *= 3; // i = i * 3;
i %= 5; // i = i % 5;
```

Operatori

- **Specijalni operatori**

`?:`	uslovni operator
`(`***`imeTipa`***`)`	eksplicitna konverzija tipa (*cast*)
`+`	konkatenacija stringova

Primeri

```
int i;
i = (int)3.14; // konverzija iz tipa double u tip int
System.out.println(i); // štampa 3

System.out.println("Novi " + "Sad"); // štampa: Novi Sad
System.out.println("Broj " + i);     // štampa: Broj 3
// Bitno je da je jedan operand tipa String, drugi će biti
// automatski konvertovan u String
```

Uslovni operator ? :

- Jedini **ternarni operator** u Javi
  - Potrebno mu je proslediti tri operanda
- Koristi se u obliku:

  `izraz1 ? izraz2 : izraz3`

  gde je `izraz1` tipa `boolean`, a ostali izrazi mogu biti bilo kog tipa (ne obavezno istog)
- Izvršava se na sledeći način:
  - Izračuna se vrednost izraza `izraz1`
  - Ako je ta vrednost `true`, vrednost celog izraza dobija se izračunavanjem vrednosti izraza `izraz2`
  - U protivnom, vrednost celog izraza dobija se izračunavanjem vrednosti izraza `izraz3`
- Izračunavanje izraza `izraz2` i `izraz3` se radi po potrebi, tj. “lenjo”

Uslovni operator ? : – primer

```
class UslovniOperator {
  public static void main(String[] args) {
    String status;
    int bodovi = 80;
    status = (bodovi >= 50) ? "Polozio" : "Nije polozio";
    System.out.println(status);
    int i = 0;
    int j = 22;
    System.out.println((i < 10) ? "Manji od 10" : j++);
    System.out.println("j = " + j);
  }
}
```

- Izlaz:

```
Polozio
Manji od 10
j = 22
```

Operatori

- **Ostali operatori**

  `instanceof` – pripadnost referencijalnom tipu

  `.` (tačka) – pristup članu klase, paketa...

  `[]` (uglaste zagrade) – pristup elementu niza

  `new` – kreiranje instance klase

- Većinu ovih operatora detaljnije ćemo obraditi kasnije

Operatori: prioritet

- Svi operatori razvrstani su po prioritetu, tako da je za svaki dobro formiran izraz tačno poznato kojim se redosledom izračunavaju vrednosti operanada
- Prioritet operatora se menja korišćenjem zagrada (i)
- Primer: vrednost izraza

```
6 % 2 * 5 + 4 / 2 + 88 - 10
```

izračunava se kao da su zagrade stavljene na sledeći način:

```
((((6 % 2) * 5) + (4 / 2)) + 88) - 10
```

Redosled izračunavanja može se potpuno promeniti:

```
6 % ((((2 * 5) + 4) / (2 + 88) - 10))
```

Operatori: prioritet

Operator	Komentar
`. [] new pozivMetoda()`	Operatori najvećeg prioriteta
`-- ++`	Postfiksni operatori
`(imeTipa) ~ ! -- ++ + -`	Unarni operatori. Prefiksni operatori
`* / %`	Množenje, deljenje, ostatak
`+ -`	Sabiranje, konkatenacija i oduzimanje
`<< >> >>>`	Pomeranje bitova
`< > <= >= instanceof`	Relacioni operatori
`== !=`	Ispitivanje jednakosti
`&`	Konjunkcija
`^`	Ekskluzivna disjunkcija
`|`	Disjunkcija
`&&`	Logička konjunkcija
`||`	Logička disjunkcija
`?:`	Uslovni operator
`= *= /= %= += -= <<= >>= >>>= &= ^= |=`	Operatori dodele imaju najmanji prioritet

Konverzija prostih tipova

- Često je u programu (tj. izrazima) potrebno “mešati” vrednosti različitih tipova podataka
  - Ovo je rađeno i u mnogim primerima koje smo videli, kao što je recimo izraz `2e13f + 7 * 9.8`
- Ista vrednost (npr. broj 7, posmatran kao broj a ne Java literal) se različito predstavlja u memoriji (na nivou bitova) u različitim brojevnim tipovima podataka
  - Kao `int`: `00000000000000000000000000000111`
  - Kao `float`: `01000000111000000000000000000000`
- S obzirom da vrednost izraza može biti samo jednog tipa, neophodno je da se vrednosti različitih tipova podataka **konvertuju**, tj. prebace iz jedne reprezentacije u drugu

Konverzija prostih tipova

- Razlikujemo dve vrste konverzije prostih tipova:
  - Proširujuće proste konverzije
  - Sužavajuće proste konverzije
- **Proširujuće proste konverzije** su:
  - iz tipa **`byte`** u tipove **`short`**, **`int`**, **`long`**, **`float`** ili **`double`**,
  - iz tipa **`short`** u tipove **`int`**, **`long`**, **`float`** ili **`double`**,
  - iz tipa **`char`** u tipove **`int`**, **`long`**, **`float`** ili **`double`**,
  - iz tipa **`int`** u tipove **`long`**, **`float`** ili **`double`**,
  - iz tipa **`long`** u tipove **`float`** ili **`double`**,
  - iz tipa **`float`** u tip **`double`**.
- Ne dolazi do gubitka informacija, jer važi sledeće:
  - tipovi su međusobno kompatibilni,
  - ciljni tip je veći od izvornog tipa

Konverzija prostih tipova

- **Sužavajuće proste konverzije** su:
  - iz tipa **`byte`** u tip **`char`**
  - iz tipa **`short`** u tipove **`byte`** ili **`char`**
  - iz tipa **`char`** u tipove **`byte`** ili **`short`**
  - iz tipa **`int`** u tipove **`byte`**, **`short`** ili **`char`**
  - iz tipa **`long`** u tipove **`byte`**, **`short`**, **`char`** ili **`int`**
  - iz tipa **`float`** u tipove **`byte`**, **`short`**, **`char`**, **`int`** ili **`long`**
  - iz tipa **`double`** u tipove **`byte`**, **`short`**, **`char`**, **`int`**, **`long`** ili **`float`**
- Može doći do gubitka informacija, pri konverziji se odseca decimalni deo (konverzija realni-celi) i viši bajtovi broja (konverzije celi-celi)
- Ove konverzije programer mora eksplicitno naznačiti korišćenjem cast operatora
  - Primer: `int n = (int)53.7;`

    (bez cast operatora kompajler bi prijavio grešku)

Konverzija prostih tipova

- Proširujuću prostu konverziju po pravilu Java kompajler može da reguliše automatski, bez intervencije programera, ali treba biti svestan njenog postojanja i pravila
  - Pogrešna očekivanja mogu dovesti do grešaka u kodu
  - Nepotrebna konverzija može usporiti program

- Primer: `2e13f + 7 * 9.8`
  - Literal `7` je tipa `int`, a literal `9.8` tipa `double`, pa je podizraz `7 * 9.8` tipa `double` i vrši se konverzija iz `int` u `double`
  - Literal `2e13f` je tipa `float`, i konvertuje se u `double` da bi tip celog izraza bio tipa `double`
  - Da je izraz bio zapisan `2e13 + 7.0 * 9.8` ne bi bilo konverzije

Konverzija prostih tipova

Pravila za automatsko unapređenje (promociju) tipova koja se primenjuju u izrazima:

- **Pravilo unarne numeričke promocije** (primena unarnih operatora)**:** vrednost tipa `byte`, `short` ili `char` se menja proširujućom prostom konverzijom u tu istu vrednost tipa `int`
- **Pravilo binarne numeričke promocije** (primena binarnih operatora)**:**
  - Ako je jedan operand tipa `double` onda se i drugi konvertuje u tip `double`
  - Inače, ako je jedan operand tipa `float` onda se i drugi konvertuje u tip `float`
  - Inače, ako je jedan operand tipa `long` onda se i drugi konvertuje u tip `long`
  - Inače, ako oba operanda već nisu tipa `int`, onda se konvertuju u tip `int`

Konverzija prostih tipova

Primer:

```
short s = 42;
s = -s;
s = s + 1;
```

- Šta nije u redu sa datim kodom?

- Zbog automatske promocije tipova, mora se upotrebiti cast operator:

```
s = (short)-s;
s = (short)(s + 1);
```

zato što su izrazi `-s` i `s + 1` promovisani u tip `int`