Osnovne Java klase

Programski jezici II

Paket java.lang

- paket java.lang – automatski se uvozi, za vrijeme kompajliranja, u svaku datoteku s izvornim kodom
- u ovom paketu nalazi se:
  - Object klasa – osnovna klasa u hijerarhiji Java klasa
  - okružujuće klase (Boolean, Character, Byte, Short, Integer, Long, Float i Double) koje predstavljaju objektnu reprezentaciju odgovarajućih primitivnih tipova
  - klase neophodne za interakciju sa JVM (Runtime),
  - upravljanje sigurnošću (SecurityManager),
  - učitavanje klasa (ClassLoader),
  - rad sa nitima (Thread, ThreadGroup),
  - izuzecima (Throwable),
  - klase koje obezbjeđuju standardni ulazni i izlazni tok, kao i tok za greške (System),
  - klase za rad sa stringovima (String, StringBuilder, StringBuffer),
  - klase za rad sa matematičkim funkcijama (Math), i dr.

Klasa Object

- Klasa Object se nalazi na vrhu hijerarhijskog stabla Java klasa
- Ako u deklaraciji klase nije navedena ključna riječ extends, onda klasa implicitno nasljeđuje Object klasu – na ovaj način sve klase nasljeđuju Object klasu, direktno ili indirektno
- Object klasa definiše osnovne funkcionalnosti koje sve klase nasljeđuju i svi objekti ispoljavaju
- Klasa Object obezbjeđuje više opštekorisnih metoda (11 metoda, finalize - deprecated)
  - 5 metoda se odnosi na konkurentno programiranje

Metoda clone

- metodom clone – kreiraju se novi objekti identični (sa identičnim stanjem) trenutnom objektu
- ovom metodom kopiraju se vrijednosti sadržanih primitivnih tipova i referenci – plitko kopiranje

```
protected Object clone() throws CloneNotSupportedException
```

- redefinisanje – u cilju obezbjeđenja vlastite implementacije – npr., kod kompozitnih objekata moguće je redefinisati metodu clone na takav način da se kloniraju i svi sadržavajući objekti – duboko kopiranje
- dobra praksa je da se redefinisana metoda deklariše kao public kako bi svakom klijentu bilo dozvoljeno da klonira objekte date klase
- ako se redefinisana clone metoda u klasi nasljednici oslanja na clone metodu Object klase, klasa nasljednica mora implementirati interfejs Cloneable kako bi označila da objekat ove klase može biti sigurno kloniran – u suprotnom, clone metoda Object klase će baciti izuzetak CloneNotSupportedException

Metoda equals

- metoda equals klase Object vraća istinitu vrijednost ako i samo ako dvije reference koje se porede referenciraju isti objekat
- metoda equeals implementira relaciju ekvivalencije za reference koje nisu null na sljedeći način:
  - refleksivnost: za referencu x (koja nije null) x.equals(x) treba da vrati istinitu vrijednost (true)
  - simetričnost: za reference x i y (koje nisu null) x.equals(y) treba da vrati istinitu vrijednost (true) ako i samo ako y.equals(x) vraća istinitu vrijednost
  - tranzitivnost: za reference x, y i z (koje nisu null), ako x.equals(y) vraća true i y.equals(z) vraća true, onda i x.equals(z) vraća true
  - konzistentnost: za reference x i y (koje nisu null), višestruki pozivi x.equals(y) treba da uvijek vrate istinitu vrijednost (true), ako informacije korištene pri poređenju nisu mijenjane
  - za svaku referencu x (koja nije null) x.equals(null) treba vratiti false

Metoda equals

- metoda equals vraća istinitu vrijednost ako i samo ako dvije reference koje se porede referenciraju isti objekat

```
public class Telefon {
	private String proizvodjac;
	private String model;

	public Telefon(String proizvodjac, String model){
		this.proizvodjac = proizvodjac;
		this.model = model;
	}

	public static void main (String[] args) {
		Telefon t1 = new Telefon("Nokia", "6300i");// 1
		Telefon t2 = new Telefon("Nokia", "6300i");// 2
		System.out.println(t1.equals(t2));         // 3
	}
}
```

Metoda equals

- metoda equals se obično redefiniše kako bi se obezbijedila semantika jednakosti vrijednosti objekata, kao što je slučaj sa klasom String i okružujućim klasama

```
public boolean equals(Object obj) {
	if(obj instanceof Telefon){
		Telefon t = (Telefon)obj;
		if(proizvodjac.equals(t.proizvodjac) &&
model.equals(t.model))
			return true;
	}
	return false;
}
```

Metoda hashCode

- vraća hash code reprezentaciju objekta
- ako se objekti smještaju u hash strukture, metoda hashCode može se koristiti za dobijanje hash vrijednosti objekta
- za ovu vrijednost se garantuje da će biti konzistentna za vrijeme izvršavanja programa, ako informacije korištene pri poređenju (metoda equals) nisu mijenjane
- ako su dva objekta jednaka prema equals metodi, njihova hash vrijednost trebala bi biti jednaka
- iz tog razloga u slučaju redefinisanja equals metode, neophodno je redefinisati i hashCode metodu
- u slučaju kada je equals metoda redefinisana, a hashCode metoda nije mogu se javiti problemi sa pretraživanjem objekata u hash strukturama

```
public int hashCode()
```

- ova metoda obično vraća memorijsku adresu objekta kao podrazumijevanu hash vrijednost objekta transformisanu u int vrijednost

Metoda hashCode

```
public static void main(String[] args) {
	Set<Telefon> telefoni = new HashSet<Telefon>();
	Telefon t1 = new Telefon("Nokia", "6300i");
	Telefon t2 = new Telefon("Nokia", "N95");
	telefoni.add(t1);
	telefoni.add(t2);
	System.out.println(telefoni.contains(new
Telefon("Nokia", "6300i")));
}
```

```
public int hashCode() {
	return proizvodjac.hashCode() + model.hashCode();
}
```

Metoda toString

- metoda toString vraća tekstualnu (string) reprezentaciju objekta

```
public String toString()
```

- ako klasa ne redefiniše toString metodu, ova metoda će vratiti tekstualnu reprezentaciju objekta u formatu: naziv klase + znak „@“ + hash vrijednost objekta

```
<naziv klase>@<hash vrijednost objekta>
```

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

- ova metoda se obično redefiniše i koristi za ispise na konzolu ili za potrebe *debug*-ovanja – pokušaj ispisa objekta (npr., na konzolu) dovešće do implicitnog poziva metode toString

Metoda getClass

- metoda getClass vraća java.lang.Class objekat koji predstavlja *runtime* klasu tekućeg objekta

```
System.out.println(nokiaE72.getClass().getCanonicalName());
System.out.println(nokiaE72.getClass().getSimpleName());
Field[] f = nokiaE72.getClass().getDeclaredFields();
for (Field field : f) {
   System.out.println(field.getName());
}
```

Okružujuće klase

- kako bi se primitivnim tipovima moglo operisati kao objektima, u java.lang paketu nalaze se okružujuće klase za svaki od primitivnih tipova – riječ je o klasama Boolean, Character, Byte, Short, Integer, Long, Float i Double koje okružuju primitivne tipove boolean, char, byte, short, int, long, float i double, respektivno
- sve okružujuće klase su final klase
- objekti ovih klasa su nepromjenljivi (eng. *immutable*), tj. vrijednost ovih objekata ne može biti promijenjena
- okružujuće klase obezbjeđuju metode za kreiranje i manipulaciju objektima koji okružuju primitivne vrijednosti, definišu korisne konstante, polja i metode za konverziju
- iako nije okružujuća klasa, Void klasa ima sličnu namjenu, jer predstavlja objektnu reprezentaciju povratnog tipa void – ova klasa se ne može instancirati, tj. nema javnog konstruktora
- u okružujuće klase se ubrajaju i AtomicInteger, AtomicLong, AtomicBoolean i AtomicReference

Okružujuće klase – kreiranje objekata

- sve okružujuće klase, osim klase Character, imaju po dva konstruktora pri čemu jedan kao argument ima odgovarajući primitivni tip, a drugi String

```
public Character (char value)
public Boolean (boolean value)
public Boolean (String s)
public Byte (byte value)
public Byte (String s) throws NumberFormatException
public Short (short value)
public Short (String s) throws NumberFormatException
public Integer(int value)
public Integer(String s) throws NumberFormatException
public Long(long value)
public Long(String s) throws NumberFormatException
public Float(float value)
public Float(String s) throws NumberFormatException
public Double(double value)
public Double(String s) throws NumberFormatException
```

- u slučaju konstruktora sa argumentom tipa String, vrši se konverzija iz stringa u vrijednost odgovarajućeg primitivnog tipa koji odgovara okružujućem tipu – ako konverziju nije moguće izvršiti desiće se NumberFormatException izuzetak

Okružujuće klase – kreiranje objekata

- drugi način kreiranja objekata okružujućih klasa jeste *box*-ing

```
Character c = 't';
Boolean b = true;
Integer i = 1;
Double d = 1.25;
Long l = 9L;
Float f = 1.2f;
Short s = (short)1;
Byte bb = (byte)1;
```

Okružujuće klase – kreiranje objekata

- treći način kreiranja objekata okružujućih klasa jeste korištenjem valueOf metode koja kao argument uzima vrijednost odgovarajućeg primitivnog tipa

```
Character c = Character.valueOf('t');
Boolean b = Boolean.valueOf(true);
Integer i = Integer.valueOf(1);
Double d = Double.valueOf(1.25);
Long l = Long.valueOf(9L);
Float f = Float.valueOf(1.2f);
Short s = Short.valueOf((short) 1);
Byte bb = Byte.valueOf((byte)1);
```

- sve okružujuće klase, osim klase Character, posjeduju još barem jednu valueOf metodu koja kao argument prima String

Okružujuće klase – korisne metode

- metode za konverziju stringova u objekte okružujućih klasa
  - svaka okružujuća klasa, osim Chatacter klase, ima i valueOf metodu koja kao argument prima String

```
Boolean b = Boolean.valueOf("true");
Integer i = Integer.valueOf("1");
Double d = Double.valueOf("1.25");
Long l = Long.valueOf("9");
Float f = Float.valueOf("1.0");
Short s = Short.valueOf("1");
Byte bb = Byte.valueOf("1");
```

```
Integer i = Integer.valueOf("1010", 2);
Integer i2 = Integer.valueOf("012", 8);
Integer i3 = Integer.valueOf("10", 10);
Integer i4 = Integer.valueOf("A", 16);
```

  - klase Integer, Long, Float i Short posjeduju još jednu preklopljenu valueOf metodu, koja kao argument prima String i bazu brojnog sistema

Okružujuće klase – korisne metode

- konverzija objekata okružujućih klasa u stringove
  - redefinisana toString metoda Object klase – u svakoj okružujućoj klasi

```
String cs = c.toString();
String bs = b.toString();
String is = i.toString();
String ds = d.toString();
String ls = l.toString();
String fs = f.toString();
String ss = s.toString();
String bbs = bb.toString();
```

Okružujuće klase – korisne metode

- Konverzija primitivnih tipova u stringove
  - svaka okružujuća klasa posjeduje i statičku toString metodu koja kao argument uzima odgovarajući primitivni tip, a kao rezultat vraća String

```
String cs = Character.toString('t');
String bs = Boolean.toString(true);
String is = Integer.toString(1);
String ds = Double.toString(1.25);
String ls = Long.toString(9);
String fs = Float.toString(1.0f);
String ss = Short.toString((short) 1);
String bbs = Byte.toString((byte) 1);
```

Okružujuće klase – korisne metode

- Konverzija objekata okružujućih klasa u primitivne tipove
  - *unbox*-ing

```
char cp = c;
boolean bp = b;
int ip = i;
double dp = d;
long lp = l;
float fp = f;
short sp = s;
byte bbp = bb;
```

  - typeValue metode

```
char cp = c.charValue();
boolean bp = b.booleanValue();
int ip = i.intValue();
double dp = d.doubleValue();
long lp = l.longValue();
float fp = f.floatValue();
short sp = s.shortValue();
byte bbp = bb.byteValue();
```

Okružujuće klase – korisne metode

- Poređenje objekata okružujućih klasa
  - Okružujuće klase implementiraju Comparable interfejs. Na ovaj način implementira se metoda compareTo. Metoda compareTo vraća 0 ako su vrijednosti primitivnih tipova, sadržane u objektima okružujućih klasa koji se porede, identične

```
Double d1 = 1.25, d2 = 1.26;
int test = d1.compareTo(d2);
System.out.println(test);
Integer i1 = 1, i2 = 1;
test = i1.compareTo(i2);
System.out.println(test);
```

```
Double d1 = 1.25, d2 = 1.26;
boolean test = d1.equals(d2);
System.out.println(test);
Integer i1 = 1, i2 = 1;
test = i1.equals(i2);
System.out.println(test);
```

  - korištenjem redefinisane metode equals iz klase Object

Okružujuće klase – korisne metode

- Konverzije stringova u numeričke vrijednosti
  - parseType metoda – u slučaju da konverziju nije moguće izvršiti desiće se izuzetak NumberFormatException

```
int i = Integer.parseInt("1");
double d = Double.parseDouble("1.25");
long l = Long.parseLong("9");
float f = Float.parseFloat("1.0");
short s = Short.parseShort("1");
byte bb = Byte.parseByte("1");
```

```
int i = Integer.parseInt("1010", 2);
long l = Long.parseLong("012", 8);
short s = Short.parseShort("10", 10);
byte bb = Byte.parseByte("a", 16);
```

  - okružujuće klase Byte, Short, Integer i Long posjeduju i odgovarajuću preklopljenu metodu koja pored String argumenta, prima i bazu brojnog sistema

Okružujuće klase – korisne metode

- Konverzija integer vrijednosti u stringove
  - okružujuće klase Integer i Long posjeduju statičke metode za konverziju integer vrijednosti u odgovarajuću string reprezentaciju u odgovarajućim notacijama (binarna, oktalna i heksadecimalna)

```
// klasa Integer
static String toBinaryString(int i)
static String toHexString(int i)
static String toOctalString(int i)
// klasa Long
static String toBinaryString(long i)
static String toHexString(long i)
static String toOctalString(long i)
```

  - može se koristiti i metoda toString

Klasa String

- klasa String iz paketa java.lang omogućava rad s nepromjenljivim (konstantnim) nizovima karaktera
- preostale dvije klase koje omogućavaju rad sa nizovima karaktera u Javi su StringBuilder i StringBuffer
- Java platforma koristi UTF-16 enkodovanje pri smještanju karaktera u nizove karaktera i objekte klasa za rad sa stringovima
- String objekti su nepromjenljivi (eng. *immutable*), što znači da se jednom kreiran objekat klase String ne može promijeniti
- sve metode klase String koje vraćaju objekat klase String, uvijek vraćaju novi objekat, dok originalni uvijek ostaje netaknut
- bitno je napomenuti i da se parametri metoda tipa String ponašaju kao da se prenose po vrijednosti, a ne po referenci, iako su u pitanju objekti, a ne primitivni tipovi

Klasa String – kreiranje objekata

- korištenjem string literala – String literal je referenca na objekat klase String, pa je iz tog razloga njima moguće manipulisati kao bilo kojom drugom String referencom

```
String s1 = "Ovo je prvi string!";
String s2 = "Ovo je drugi string!";
```

```
int duzina = "Ovo je prvi string!".length();
int hash = "Ovo je drugi string!".hashCode();
```

- kompajler optimizuje rukovanje string literalima – svi string literali sa identičnom sekvencom karaktera dijele samo jedan objekat klase String – pored toga, i konstantni izrazi koji se izračunavaju za vrijeme kompajliranja, a koji rezultiraju identičnom sekvencom karaktera, dijeliće identičan objekat klase String
- String Literal Pool

```
String s1 = "Ovo je prvi string!";
String s2 = "Ovo je prvi string!";
String s3 = "Ovo je " + " prvi string!";

String temp = "Ovo je ";
String s4 = temp + " prvi string!";
```

- s4 neće dijeliti isti String objekat – ne izračunava se za vrijeme kompajliranja

Klasa String – kreiranje objekata

- String Literal Pool – kolekcija referenci na String objekte
- String objekti se čuvaju na heap-u, kao i svaki drugi objekti

- metoda intern()
  - vraća kanoničku reprezentaciju string objekta

- pool stringova je inicijalno prazan
- održava se “privatno”, od strane klase String
- pri pozivu metode intern, ako pool sadrži string jednak String objektu (prema metodi equals), onda se vraća string iz pool-a – u suprotnom, String objekat se dodaje u pool i referenca na taj objekat se vraća
- odatle, za svaka dva stringa s i t, s.intern() == t.intern() je istinito ako i samo ako je s.equals(t) istinito

Klasa String – kreiranje objekata

```
public class StringTest {
        public static void main(String[] args) {
                String string = "string";
                String s1 = "test string";
                String s2 = "test string";
                String s3 = "test" + " string";
                String s4 = "test" + " " + string;
                String s5 = new String("test string");
                String s6 = ("test " + string).intern();
                System.out.println(s1==s2);
                System.out.println(s1==s3);
                System.out.println(s1==s4);
                System.out.println(s1==s5);
                System.out.println(s1==s6);
                }
}
```

Klasa String – kreiranje objekata

- klasa String ima 15 konstruktora sa različitim argumentima (2 *deprecated*)
- objekti klase String kreirani korištenjem konstruktora su uvijek novi objekti, bez obzira na sekvencu karaktera koju sadrže

```
String s1 = new String("String");          // 1
char[] ch = {'S', 't', 'r', 'i', 'n', 'g'};
String s2 = new String(ch);                // 2
byte[] by = {83, 116, 114, 105, 110, 103};
String s3 = new String(by);                // 3
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
```

Klasa String – korisne metode

- metode za pretraživanje karaktera i podstringova

```
int indexOf(int ch)                          // 1
int indexOf(int ch, int fromIndex)           // 2
int indexOf(String str)                      // 3
int indexOf(String str, int fromIndex)       // 4
int lastIndexOf(int ch)                      // 5
int lastIndexOf(int ch, int fromIndex)       // 6
int lastIndexOf(String str)                  // 7
int lastIndexOf(String str, int fromIndex)// 8
boolean contains(CharSequence s)             // 9
```

Klasa String – korisne metode

- metode za poređenje stringova i dijelova stringova

```
boolean endsWith(String suffix)                    // 1
boolean startsWith(String prefix)                  // 2
boolean startsWith(String prefix, int offset)      // 3
int compareTo(String anotherString)                // 4
int compareToIgnoreCase(String str)                // 5
boolean equals(Object anObject)                    // 6
boolean equalsIgnoreCase(String anotherString)     // 7
boolean regionMatches(int toffset, String other, int
ooffset, int len)                                  // 8
boolean regionMatches(boolean ignoreCase, int toffset,
String other, int ooffset, int len)                // 9
boolean matches(String regex)                      // 10
```

Klasa String – korisne metode

- metode za zamjenu karaktera i podstringova

```
String replace(char oldChar, char newChar)
String replace(CharSequence target, CharSequence
replacement)
String replaceAll(String regex, String replacement)
String replaceFirst(String regex, String replacement)
```

- metode za ekstrakciju podstringova

```
String substring(int beginIndex)
String substring(int beginIndex, int endIndex)
String trim()
```

Klasa String – korisne metode

- metode za promjenu veličine slova

```
String toLowerCase()
String toLowerCase(Locale locale)
String toUpperCase()
String toUpperCase(Locale locale)
```

- metode za određivanje dužine stringa

```
int length()
boolean isEmpty()
```

- metode za čitanje karaktera stringa

```
char charAt(int index)                                          // 1
void getChars(int srcBegin, int srcEnd, char[] dst, int
dstBegin)                                                       // 2
```

Klasa String – korisne metode

- metode za konkatenaciju stringova

```
String concat(String str)
```

- metode za konverziju vrijednosti različitih tipova u stringove

```
static String valueOf(Object obj)
static String valueOf(char[] charArray)
static String valueOf(boolean b)
static String valueOf(char c)
static String valueOf(int i)
static String valueOf(long l)
static String valueOf(float f)
static String valueOf(double d)
```

Klasa String – korisne metode

- metode za formatiranje stringova

```
static String format(String format, Object... args)
static String format(Locale l, String format,
Object... args)
```

- metode za rad sa regularnim izrazima

```
boolean matches(String regexStr)
String replaceAll(String regex, String replacement)
String replaceFirst(String regex, String replacement)
String[] split(String regexStr, int limit)
String[] split(String regexStr)
```

Klase StringBuilder i StringBuffer

- StringBuilder (od JDK 1.5) i StringBuffer (od JDK 1.0) klase implementiraju promjenljive sekvence karaktera
- pored mogućnosti promjene sekvence karaktera koje sadrže objekti ovih klasa, i sam kapacitet dinamički može biti promijenjen
- Iako su objekti ove dvije klase i objekti klase String bliski, riječ je o nezavisnim final klasama koje nemaju roditeljsku klasu, tj. koje implicitno nasljeđuju Object klasu
  - iz ovog razloga StringBuilder i StringBuffer reference se ne mogu dovesti u relaciju sa String referencama, čak ni eksplicitnim kastovanjem, i obrnuto

Klase StringBuilder i StringBuffer

- klase StringBuilder i StringBuffer su identične, osim što StringBuilder nije sinhronizovana
- ovaj nedostatak StringBuilder klase može dovesti do neželjenih posljedica u slučaju kada više programskih niti istovremeno pristupaju objektu ove klase
- važno: i objekat klase String je siguran u slučaju istovremenog pristupa putem više programskih niti, jer je nepromjenljiv
- korištenje klase StringBuilder se preporučuje u situacijama kada se očekuje modifikacija sekvence karaktera samo putem jedne programske niti
- korištenje klase StringBuffer se preporučuje u situacijama kada se očekuje modifikacija sekvence karaktera putem više programskih niti
- StringBuilder implementacija je brža