Razvoj softvera

dr.sc. Emir Mešković

III predavanje

Sadržaj predavanja

- Naslijeđivanje
- `Object` klasa
- Generička `ArrayList`
- `Number` i `Random` klase

Nasljeđivanje

- Nasljeđivanje omogućava kreiranje novih klasa koje se grade na osnovu postojećih.
- Kada klasa (*subklasa* ili *izvedena* klasa) naslijedi od postojeće klase (*topklasa, supeklasa ili bazna*), subklasa sadrži sve metode i članove od topklase i može definirati proizvoljan broj novih metoda i članova za nove situacije u kojima će se koristiti objekti izvedene klase.
- Nasljeđivanjem se ostvaruje relacija **je,** tj objekat tipa subklase ujedno **je** i tipa topklase i može se koristiti bilo gdje gdje se očekuje objekat bazna klasa. (radnik **je** čovjek, golub **je** ptica itd.)
- Za definiranje subklasa Java koristi ključnu riječ **`extends`**

extends

Java

```
class Izvedena extends Bazna
{
    //Implementacija
}
```

C++

```
class Izvedena : public Bazna
{
    //Implementacija
};
```

- Java subklasa može nasljeđivati samo od jedne klase (c++ klase mogu biti izvedene od više klasa)
- Java podržava samo javno (public) nasljeđivanje (c++ obezbijeđuje još i private i protected)
- Metodi i članovi nasljeđuju se na način:
  - Privatni članovi bazne klase iako postoje u izvedenoj klasi nisu direktno dostupni metodima izvedene klase
  - Public i protected članovi bazne klase dostupni su u metodima izvedene klase.
  - Statički članovi bazne klase nasljeđeni su u izvedenoj klasi na način da bazna i izvedena klasa dijele zajedničku kopiju statičkih članova.

Konstruktori subklasa


```
class Izvedena extends Bazna
{
    //...
    public Izvedena(...)
    {
       super(...);
       //Implementacija
    }
    //...
}
```

- Radi konstruisanja dijela objekta koji je naslijeđen od bazne klase, konstruktor izvedene klase može (**treba**) pozivati neki od konstruktora bazne klase pomoću ključne riječ `super`.
- Ako se izostavi poziv konstruktora bazne klase, onda kompajler ubacuje poziv default konstruktora bazne klase ako postoji, u suprotnom kod nije moguće iskompajlirati.

Redefinisanje metoda


```
class Izvedena extends Bazna
{
    //...
    public int racunaj(int k)
    {
       return super.racunaj(k) + 10;
    }
}
```

- Bilo koji javni metod bazne klase može se redefinisati (override) u izvedenoj klasi, pod uslovom da se u izvedenoj klasi implementira metod sa istim potpisom (isto ime i isti ulazni parametri)
- Ukoliko je potrebno pozvati metod kao što je definisan u baznoj klasi koristi se prije poziva metoda ključna riječ `super`.

Vezivanje metoda

- C++ sve metode po default-u veže statički
  - tj verzija metoda koja će se pozvati na objektu ovisi od njegovog deklariranog tipa i određuje se prilikom kompajliranja
  - Da bi neki metod bio vezan dinamički, metod mora biti eksplicitno u baznoj klasi označen kao virtual.

```
#include <iostream>
class Bazna
{
  public:
    void ispis1() { std::cout << "Bazna ispis1\n"; }
    virtual void ispis2() { std::cout << "Bazna ispis2\n"; }
};
class Izvedena : public Bazna
{
  public:
    void ispis1() { std::cout << "Izvedena ispis1\n";}
    void ispis2() { std::cout << "Izvedena ispis2\n";}
};
int main()
{
    Bazna* i = new Izvedena();
    i->ispis1();
    i->ispis2();
    delete i;
}
```

Šta je ispis ovog programa?

Java klase i vezivanje

- Java veže:
  - `private` i `static` metode kao i sve konstruktore statički
  - a, `public` metode dinamički
    - Da bi `public` metod bio vezan statički potrebno ga je označiti kao `final`
- Modifikator `final` ima različita značenja ovisno od konteksta njegove upotrebe
  - `final` za klase, onemogućuje kreiranje izvedenih klasa od klase označene kao `final`. Sljedeće bi izazvalo grešku:
    - `final class MojString {};`
    - `class String1 extends Mojstring {};`
  - `final` za metode, onemogućava redefinisanje javnog metoda označenog kao `final` u izvedenim klasama
  - `final` za članove, onemogućava promjenu člana označenog kao `final` nakon izvršenja konstruktora klase.

Klase i cast operator

- Konverzija reference objekta izvedene klase u baznu klasu (upcast) vrši se implicitno:
  - `Bazna b = new Izvedena();`
    - Uvijek u redu jer `Izvedena` je ujedno i `Bazna`.
- Konverzija u obrnutom smjeru (downcast) mora biti eksplicitna tj izvršena pomoću cast operatora:

```
Bazna[] b = new Bazna[2];
b[0] = new Izvedena1();
b[1] = new Izvedena2();
Izvedena1 a = (Izvedena1) b[0]; // u redu
Izvedena1 c = (Izvedena1) b[1]; // runtime greska
```

Klase i cast operator

- Ako konverzija ne uspije (kao u primjeru za objekat c) generira se greška pri izvršavanju. Pripadnost određenoj klasi može se provjeriti pomoću `instanceof` operatora npr:

```
Izvedena1 c;
if (b[1] instanceof Izvedena1)
c = (Izvedena1) b[1];
```

- Konverziju između klasa moguće raditi samo ako pripadaju istoj nasljednoj hijerarhiji

Object klasa

- Klasa Object, definirana u paketu java.lang, predstavlja klasu od koje nasljeđuju sve Java klase
  - jedina Java klasa koja nema svoju topklasu
- Ako klasa nema roditelja eksplicitno, tada je ta klasa direktna subklasa klase Object tj sljedeće definicije su iste
  - `class MojaKlasa {}`
  - `class MojaKlasa extends Object {}`
- Iz ovog proizilazi da se referenca bilo koje klase, može implicitno konvertirati u Object referencu:

```
class Test1 {...}
class Test2 {...}
class Test
{
    public static void main(String[] args)
    {
        Object[] obj = new Object[2];
        obj[0] = new Test1();
        obj[1] = new Test2();
    }
}
```

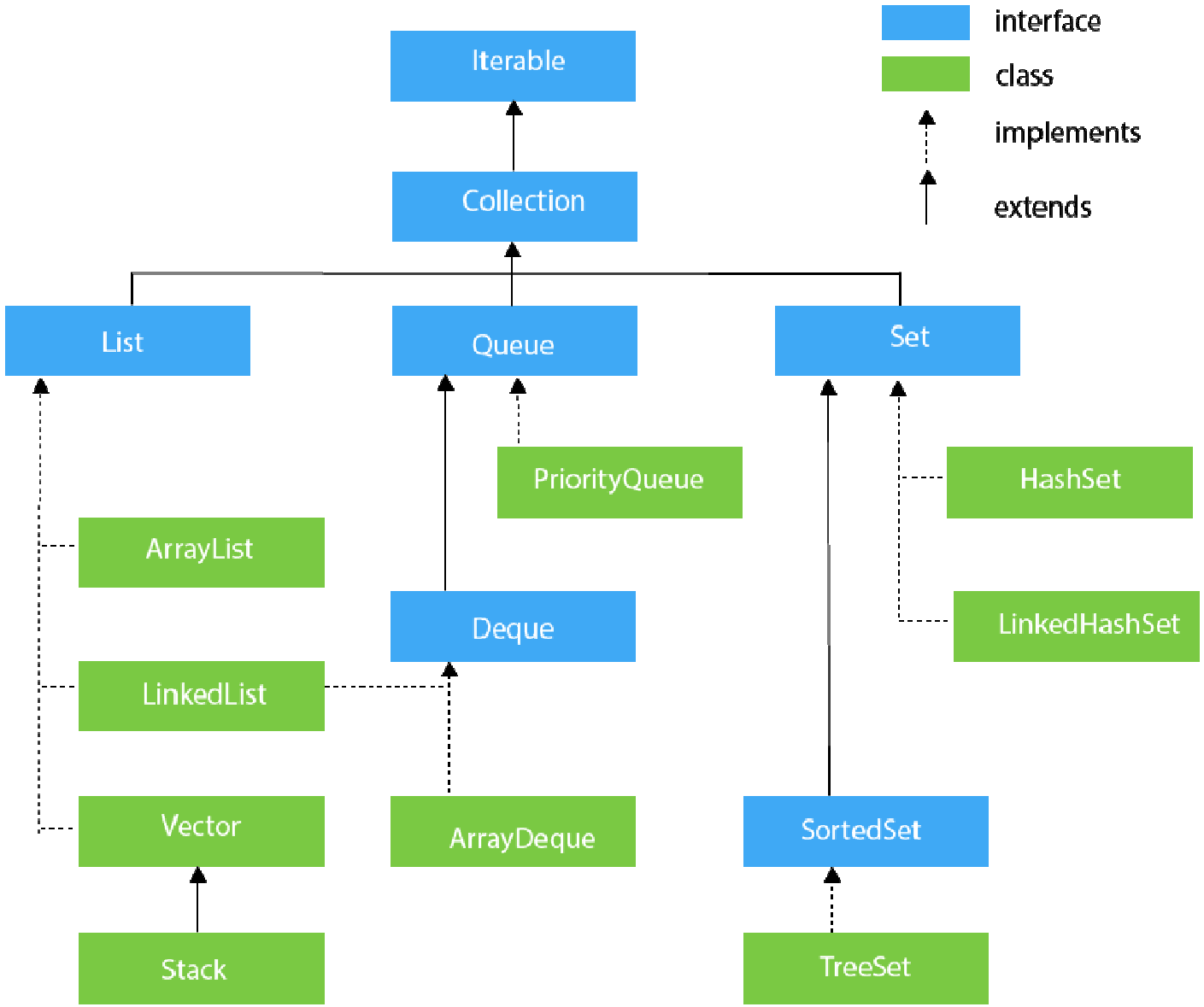
`Object` metodi

- Neki metodi klase `Object`
  - `String toString()`
    - Vraća string reprezentaciju objekta
    - Trebala bi redefinisati nova klasa
  - `boolean equals(Object drugi)`
    - Vraća tačno ako `this` i drugi pokazuju na istu lokaciju u memoriji
    - Trebala bi redefinisati nova klasa
  - `Class getClass()`
    - Vraća objekat tipa Class koji sadrži informacije o klasi (ime klase, metodi klase itd)
  - `Object clone()`
    - Vraća kopiju objekta, JVM alocira memoriju kopira sadržaj objekta this (kopiraju se svi članovi klase) i
    - vraća referencu na lokaciju sa kopijom.

Primjer


```
public class Tacka
{
    private int x_, y_;
    public Tacka(int x, int y)
    {
       x_ = x;
       y_ = y;
    }
    public boolean equals(Object druga)
    {
       if(druga instanceof Tacka){
           Tacka b = (Tacka)druga;
           if((x_ == b.x_) && (y_ == b.y_))
               return true;
       }
       return false;
    }
    public String toString()
    {
       return "Tacka[x_=" + x_ + ",y_=" + y_ + "]";
    }
    public static void main(String[] args)
    {
       Tacka a = new Tacka(5,19);
       Tacka b = new Tacka(5,19);
       if(a.equals(b))
           System.out.println("Tacke na istoj lokaciji");
        System.out.println("a = " + a);
    }
}
```

Kolekcije u Javi



Generičke Array liste

- Ograničenja kod upotrebe nizova
  - Veličina je fiksna prilikom kreiranja i ne mogu se povećavati ili smanjivati nakon inicijalizacije
  - Moraju se ručno kreirati metodi za manipulaciju sadržajem npr. Dodavanje ili brisanje elementa
- `ArrayList` klasa je slična nizu, ali automatski prilagođava veličinu prilikom dodavanja ili uklanjanja elemenata
- `ArrayList` je generička klasa sa tipom kao parametrom, npr. `ArrayList<Tacka>`
  - `ArrayList<Tacka> tacke = new ArrayList<Tacka>();`
- `ArrayList` može sadržavati samo objekte, ne i primitivne tipove
- Mora se importovati `java.util.ArrayList`

Metode ArrayList

- Elementima `ArrayList` se ne pristupa korištenjem indeksa, nego upotrebom dostupnih metoda

Metod	Namjena
`add(value)`	Dodaje vrijednost na kraj liste
`add(index, value)`	Ubacuje vrijednost ispred datog indeksa, pomjerajući naredne vrijednosti udesno
`clear()`	Uklanja sve elemente iz liste
`indexOf(value)`	Vraća prvi indeks gdje je data vrijednost pronađena u listi (-1 ukoliko nije pronađeno)
`get(index)`	Vraća vrijednost na datom indeksu
`remove(index)`	Uklanja vrijednost na datom indeksu, pomjerajući naredne vrijednosti ulijevo
`set(index, value)`	Mijenja vrijednost na datom indeksu sa datom vrijednošću
`size()`	Vraća broj elemenata u listi
`toString()`	Vraća string reprezentaciju liste kao “[3, 42, -7, 15]”

Upotreba ArrayList

- Kroz `ArrayList` se može kretati na tri načina:
  - Korištenjem `for-each` petlje
  - Korištenjem iteratora (`Iterator`)
  - Korištenjem iteratora liste (`ListIterator`)
- `for-each` petlja se koristi kao kod nizova, gdje varijabla za iteraciju predstavlja konkretan član liste

```
for(String i : imena)
    System.out.println("Ime je " + i);
```

- `Iterator` je član collections framework-a
  - Metode: `hasNext(), next(), remove()`
  - Koristi se samo za prolazak unaprijed
  - Mora se importovati `java.util.Iterator`

```
Iterator<String> iterator = imena.iterator();
while(iterator.hasNext())
    System.out.println("Ime je " + iterator.next());
```

Upotreba ArrayList

- `ListIterator` je također član collections framework-a
- Omogućava prolazak kroz `ArrayList` u oba smjera
- Ne sadrži `remove()` metod
- Mora se importovati `java.util.ListIterator`

```
ListIterator<String> liter = imena.listIterator();
System.out.println("Prolaz unaprijed:");
while(liter.hasNext())
    System.out.println("Ime je " + liter.next());
System.out.println("Prolaz unazad:");
while(liter.hasPrevious())
    System.out.println("Ime je " + liter.previous());
```

Number klasa

- Ukoliko je potrebno primitivnu varijablu predstaviti kao klasu tada se koriste tzv. wrapper klase iz `java.lang` paketa
- Wrapper klasa enkapsulira ili omotova (wrap) primitivni tip unutar objekta i definirana je za svaki primitivni tip tj:

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

Konverzija primitiv/wrapper

- Java ima osobine pod nazivom AutoBoxing i Unboxing
  - AutoBoxing – automatska konverzija između primitivnih tipova i odgovarajućih wrapper klasa, npr. `Double rez = 18.85;`
  - Unboxing – konvertovanje objekta wrapper tipa u odgovarajuću primitivnu vrijednost, npr. `double rez1 = rez;`
- Vrijednost je moguće vratiti u primitivni tip i pomoću getter metoda sa imenom `tipValue()`, npr.

```
int a = rez.intValue();
double b = rez.doubleValue();
```

- AutoBoxing i Unboxing rade čak i sa aritmetičkim izrazima, npr.
  - `Integer n = 3; n++;`
- Treba voditi računa kod upotrebe operatora jednakosti (==)
- Wrapper klase posjeduju statički metod za konverziju stringa u primitivnu vrijednost, npr.
  - `Integer.parseInt("3");`

Random klasa

- Često postoje zadaci zasnovani na nekoj slučajno generisanoj vrijednosti
  - izvlačenje karte iz špila, izvlačnje brojeva binga
- Slučajni brojevi se mogu generisati pomoću metoda `random()` klase `Math`
  - generiše se `double` vrijednost u opsegu 0.0 do 1.0 (isključivo)
- Java u okviru paketa `java.util` obezbjeđuje klasu `Random` za generisanje slučajnih vrijednosti
  - sadrži metode koji slučajno generišu integer, `double`, `boolean`, `float` i `long` vrijednosti
- Kreira se objekat klase
  - `Random slucajan = new Random();`

Random klasa

- Metode za generisanje slučajnih vrijednosti
  - `boolean nextBoolean();` - generiše `true` ili `false` vrijednosti
  - `int nextInt();` - cjelobrojna vrijednost između `Integer.MIN_VALUE` i `Integer.MAX_VALUE`
  - `long nextLong();` - cjelobrojna vrijednost između `Long.MIN_VALUE` i `Long.MAX_VALUE`
  - `float nextFloat();` - decimalni broj između 0.0 i 1.0 (isključivo)
  - `double nextDouble();` - decimalni broj između 0.0 i 1.0 (isključivo)
  - `int nextInt(int maxValue);` - pozitivni brojevi od 0 do `maxValue` (isključivo)
- Ukoliko se želi generisati ista sekvenca objekat se kreira sa konstantnim cijelim brojem tipa `long`
  - `Random slucajan = new Random(20L);`