Java Web Development

Termin 06

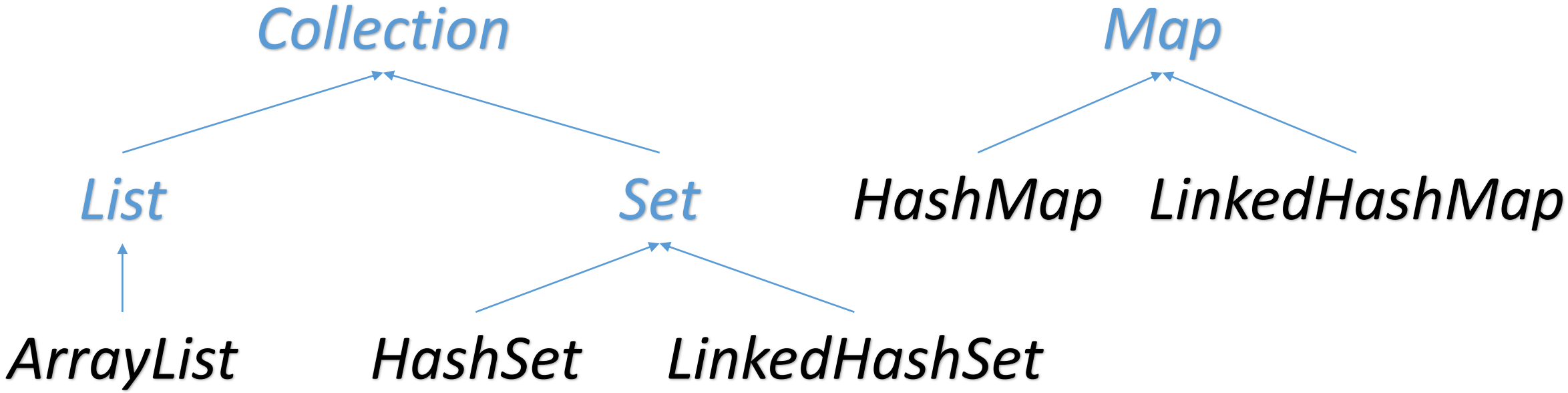
Sadržaj:

1. Osvrt na kolekcije
2. Mape
3. Iterator
4. Izuzeci
5. *Third-party* biblioteke

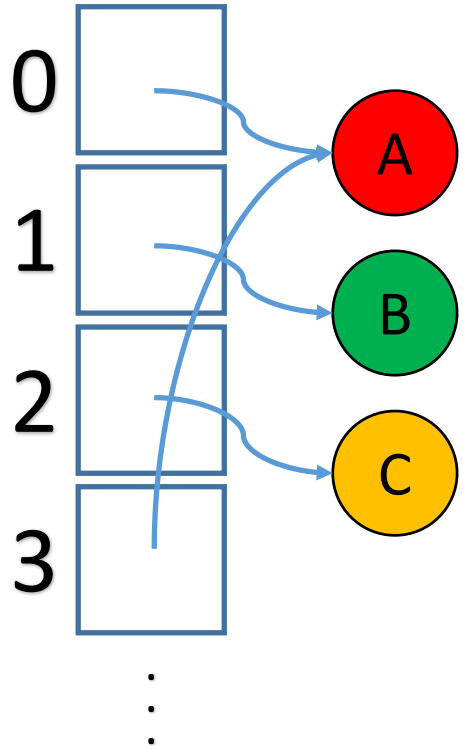
6. Kreiranje *.jar* datoteke
7. Dodatni materijali
8. Vežbanje

Kolekcije

- strukture podataka za skladištenje i manipulaciju nad grupom objekata
- ima ih mnogo, a neke od najopštijih su:



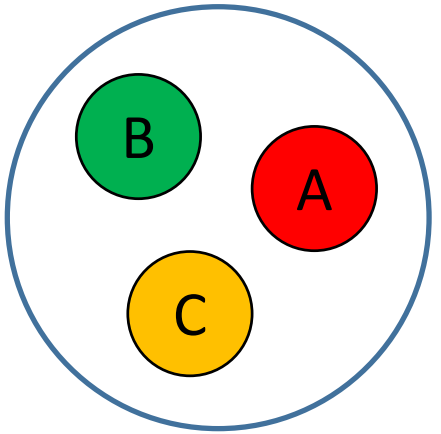
Liste



Metoda	Funkcionalnost	Napomena
iterator()	omogućuje kretanje kroz elemente	
add(Object o)	dodavanje	na kraj
contains(Object o)	provera postojanja (po vrednosti)	
remove(Object o)	uklanjanje (po vrednosti)	
add(int index, Object o)	dodavanje po indeksu (rednom broju)	
get(int index)	pristup po indeksu (rednom broju)	
remove(int index)	uklanjanje po indeksu (rednom broju)	

- moguće sortiranje
- iteracija se uvek izvodi u definisanom redosledu

Set-ovi (skupovi)



Metoda	Funkcionalnost	Napomena
iterator()	omogućuje kretanje kroz elemente	
add(Object o)	dodavanje	ne dozvoljava duplikate
contains(Object o)	provera postojanja (po vrednosti)	efikasna
remove(Object o)	uklanjanje (po vrednosti)	efikasna

- nije moguće sortiranje
- iteracija se u opštem slučaju izvodi u nasumičnom redosledu

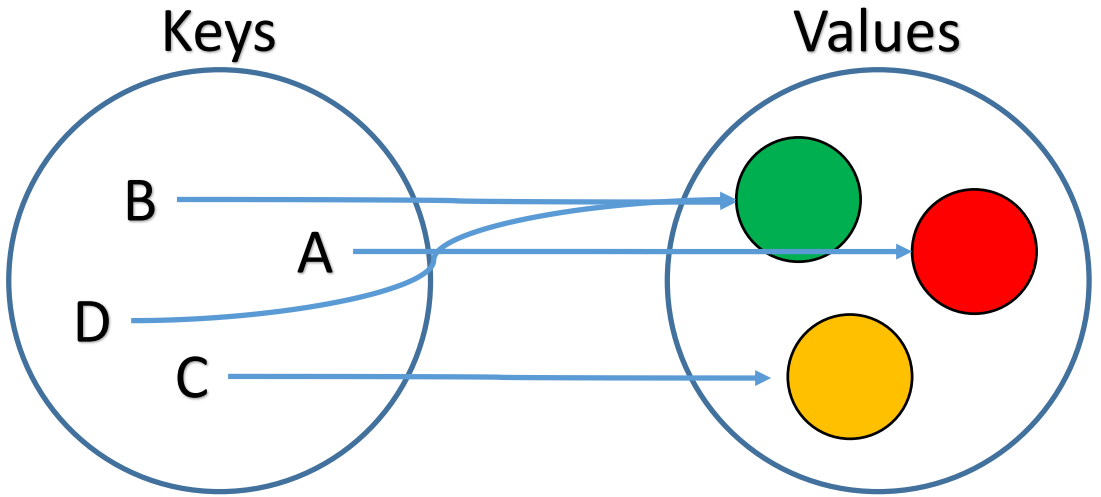
1. Osvrt na kolekcije
2. Mape
3. Iterator
4. Izuzeci
5. *Third-party* biblioteke

6. Kreiranje *.jar* datoteke
7. Dodatni materijali
8. Vežbanje

Mape

- asocijativne strukture podataka, kao i *Set*-ovi (organizuju se naspram svog sadržaja)
- za razliku od ostalih kolekcija, manipulišu parovima (ključ, vrednost)
- i ključevi i vrednosti mogu biti primitivni, složeni ili korisnički definisani tipovi
- vrednost se u mapu dodaje pod njenim jedinstvenim ključem
- uz pomoć ključa se vrši pristup vrednosti i njeno uklanjanje
- ni ključ a ni vrednost ne mogu da postoje u mapi samostalno

Mape



Metoda	Funkcionalnost	Napomena
put(Object key, Object value)	dodavanje para (ključ, element)	ne dozvoljava duplikate ključa
containsKey(Object key)	provera postojanja po vrednosti ključa	efikasna
get(Object key)	pristup po vrednosti ključa	efikasna
remove(Object key)	uklanjanje (po vrednosti ključa)	efikasna
keySet()	set ključeva	
values()	kolekcija vrednosti	

- nije moguće sortiranje
- iteracija se u opštem slučaju izvodi u nasumičnom redosledu

Mape

Dodavanje:

```
Map<String, Student> studenti = new HashMap<>();
studenti.put("0001", new Student("0001", "A", "A"));
studenti.put("0002", new Student("0002", "B", "B"));
studenti.put("0003", new Student("0003", "C", "C"));
```

Pristup:

```
System.out.print("Unesite indeks studenta: ");
String indeks = in.nextLine();

Student pronadjeniStudent = studenti.get(indeks);
if (pronadjeniStudent != null) {
    System.out.print("Pronađeni student: ");
    System.out.println(pronadjeniStudent);
} else {
    System.out.println("Student nije pronađen!");
}
```

Uklanjanje:

```
System.out.print("Unesite indeks studenta: ");
String indeks = in.nextLine();

Student uklonjeniStudent = studenti.remove(indeks);
if (uklonjeniStudent == null) {
    System.out.println("Student nije pronađen!");
}
```

1. Osvrt na kolekcije
2. Mape
3. Iterator
4. Izuzeci
5. *Third-party* biblioteke

6. Kreiranje *.jar* datoteke
7. Dodatni materijali
8. Vežbanje

Iterator

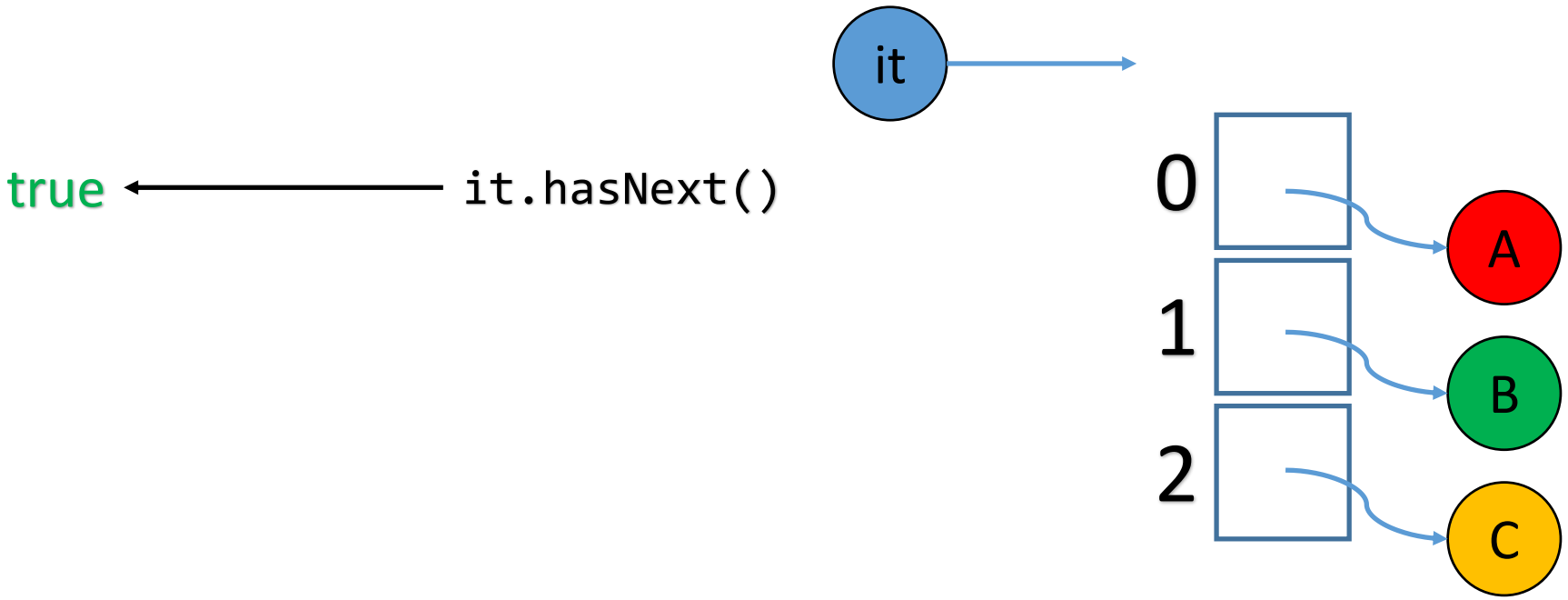
- pomoćna klasa koja omogućuje kretanje kroz elemente kolekcije
- koristi se na isti način nezavisno od odabira kolekcije (osim za mape)
- *for-each* petlja se oslanja na ovaj mehanizam

```
Iterator<Student> it = studenti.iterator();
while (it.hasNext()) {
    Student itStudent = it.next();
    System.out.println(itStudent);
}
```

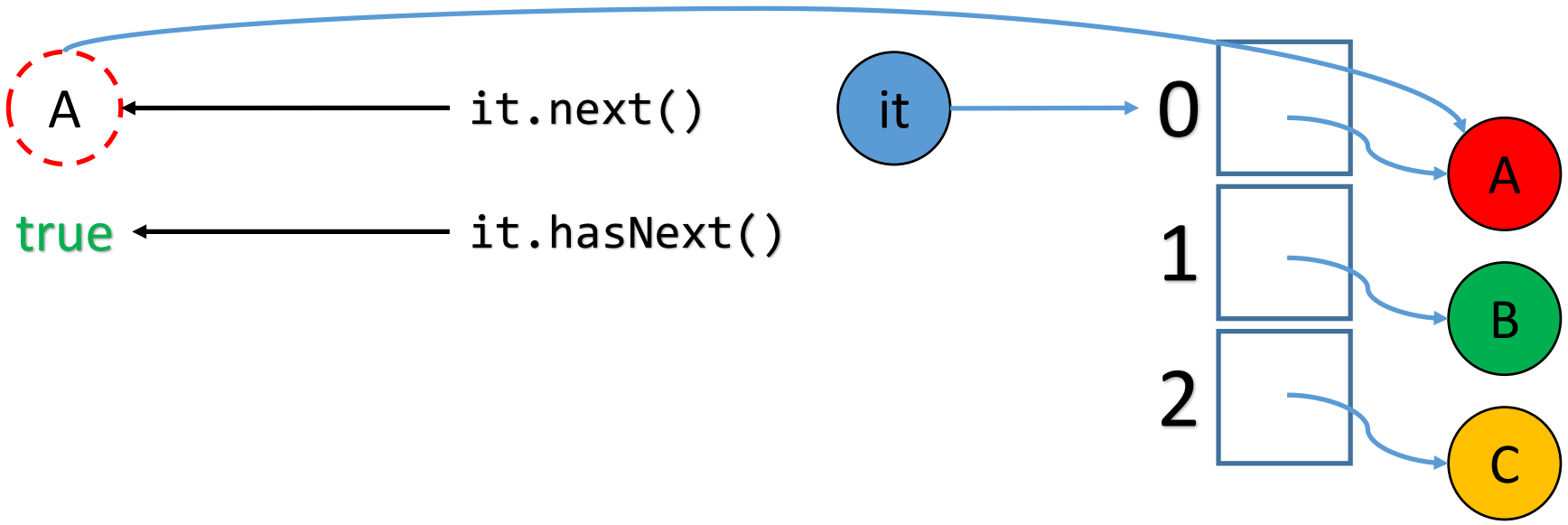
=

```
for (Student itStudent: studenti) {
    System.out.println(itStudent);
}
```

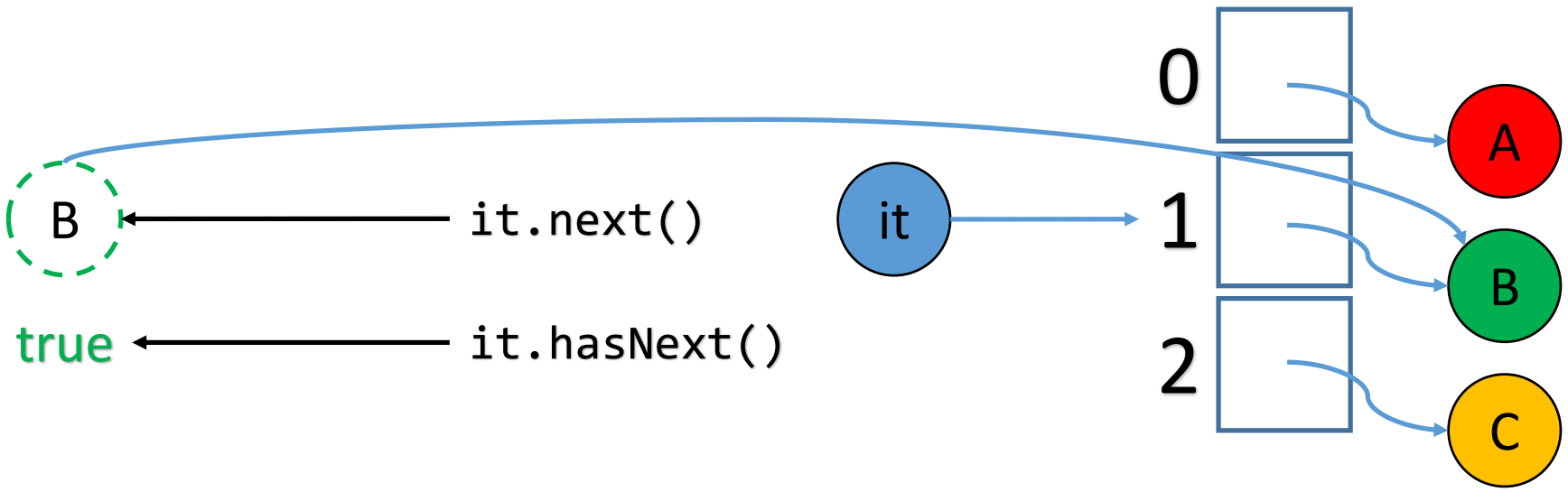
Iterator



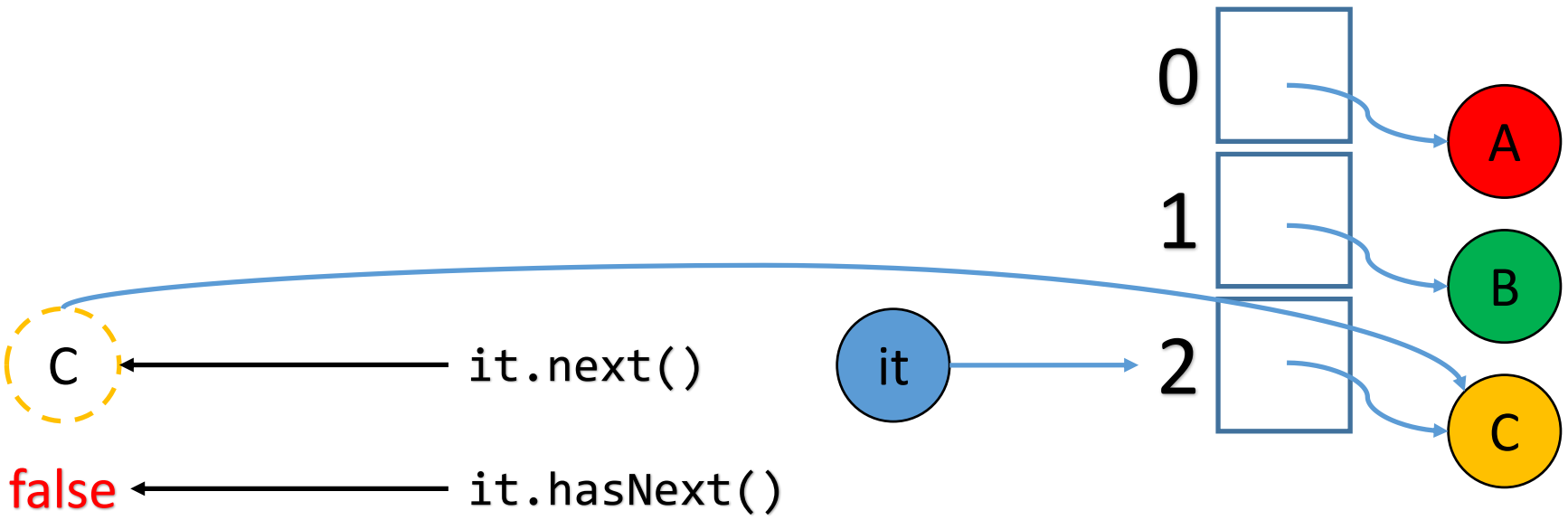
Iterator



Iterator



Iterator



Primer 1

- A_DemoList
- B_DemoSet
- C_DemoMap
- D_DemoIterator

1. Osvrt na kolekcije
2. Mape
3. Iterator
4. Izuzeci
5. *Third-party* biblioteke

6. Kreiranje *.jar* datoteke
7. Dodatni materijali
8. Vežbanje

Izuzeci (*Exceptions*)

- mehanizam koji omogućuje da se spreči zaustavljanje programa u slučaju nastanka greške u nepredviđenim i predviđenim okolnostima (događajima), da se identifikuje ta greška i da se nastavi izvršavanje programa ukoliko je to moguće, odnosno ima smisla

- *tzv. stack trace* u slučaju nastanka greške može da izgleda ovako:

```
Exception in thread "main" java.lang.NullPointerException
        at com.ftninformatika.termin06.primer01.Application.funkcija3(Application.java:16)
        at com.ftninformatika.termin06.primer01.Application.funkcija2(Application.java:10)
        at com.ftninformatika.termin06.primer01.Application.funkcija1(Application.java:6)
        at com.ftninformatika.termin06.primer01.Application.main(Application.java:21)
```

- informacija o nastaloj greški je stigla do virtualne mašine, program se zaustavlja, a svi podaci koji su bili u memoriji programa se gube

Izuzeci (*Exceptions*)

- pod predviđenim događajima se smatra ograničeni broj situacija u kojima neki kod (pored uspešnog) može da proizvede i neuspešan rezultat
- primeri predviđenih događaja su:
  - datoteka ili postoji ili ne postoji za vreme otvaranja
  - konekcija sa bazom podataka je ostvarena ili nije i sl.
- izuzeci koji opisuju ovakve događaje se zovu *Checked Exceptions*
- neki od njih su: *FileNotFoundException*, *IOException*, *SQLException* i sl.
- prevodilac obavezuje programera da kodira naredbe za rukovanje ovakvim izuzecima

Izuzeci (*Exceptions*)

- nepredviđeni događaji zavise od podataka sa kojima program radi za vreme svog izvršavanja
- oni su posledica načina korišćenja programa, kao što je npr. korisnički unos
- primeri nepredviđenih događaja su: pristup *null* referenci, pristup poziciji u listi koja ne postoji, pokušaj konverzije neodgovarajućeg *string*-a u broj, deljenje sa nulom i sl.
- izuzeci koji opisuju ovakve događaje se zovu *Unchecked Exceptions*
- neki od njih su: *NullPointerException*, *IndexOutOfBoundsExceptions*, *NumberFormatException*, *ArithmeticException* i sl.
- programer nije u obavezi da kodira naredbe za rukovanjem ovakvim izuzecima, a može

Izuzeci (*Exceptions*)

- postoji posebna grupa nepredviđenih događaja koji ne zavise ni od načina pisanja, a ni korišćenja programa, već od uslova u kojima se program izvršava kao što su operativni sistem, hardver i sl.
- klase koje opisuju ovakve događaje spadaju u grupu grešaka (*Errors*) i proizvodi ih virtualna mašina
- primeri ovakvih grešaka: *VirtualMachineError*, *OutOfMemoryError*, *StackOverflowError* i sl.
- programer ne bi trebalo da kodira naredbe za rukovanjem ovakvim izuzecima, a može

Izuzeci (*Exceptions*)

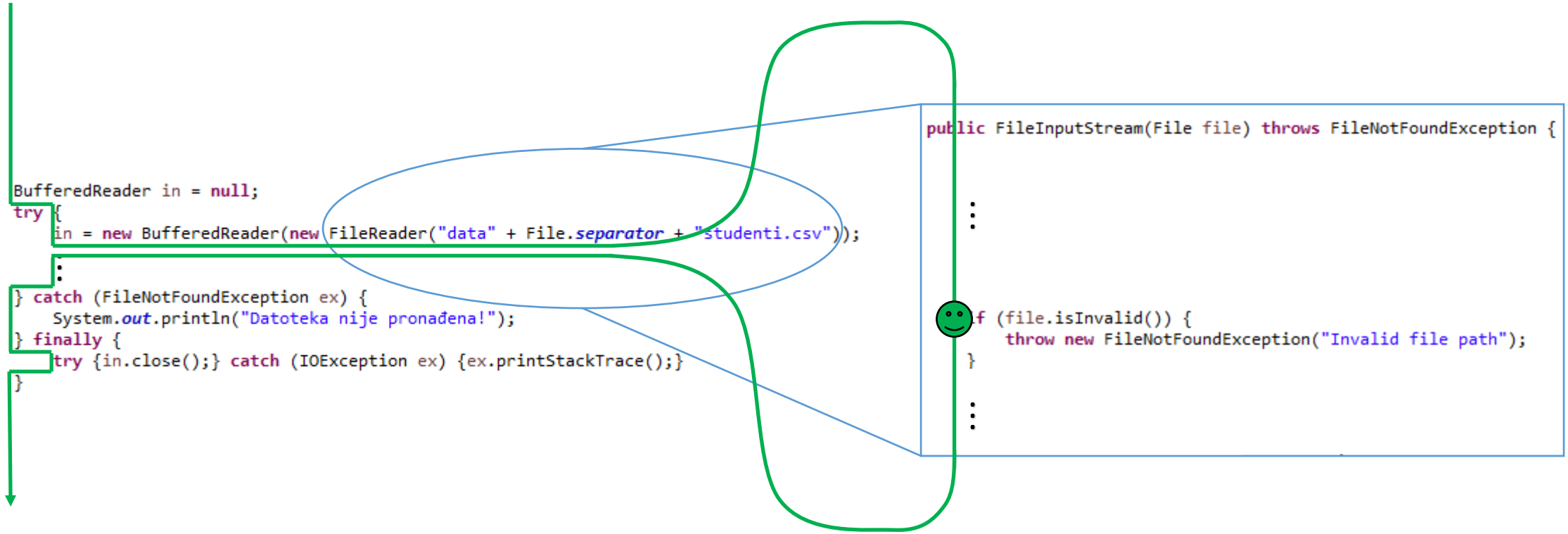
- *throws* klauzula obaveštava korisnika funkcije i prevodioca da ona može da proizvede izuzetak
- *throw* naredba u toj funkciji prekida funkciju i "ispaljuje" izuzetak (obavezno stoji u if naredbi)
- *try* blok obuhvata korišćenu funkciju i "hvata" izuzetak, odnosno sprečava da on zaustavi program
- *catch* blok definiše proceduru za obradu izuzetka
- *finally* blok definiše proceduru koja treba da se izvrši i u slučaju uspešnog i u slučaju neuspešnog izvršavanja funkcije (čak i ukoliko u try ili catch blokovima stoji return naredba)
- mora da postoji *try* blok i bar jedan od blokova *catch* ili *finally*, a mogu i oba

```
BufferedReader in = null;
try {
    in = new BufferedReader(new FileReader("data" + File.separator + "studenti.csv"));
    ⋮
} catch (FileNotFoundException ex) {
    System.out.println("Datoteka nije pronađena!");
} finally {
    try {in.close();} catch (IOException ex) {ex.printStackTrace();}
}
```

```
public FileInputStream(File file) throws FileNotFoundException {

    ⋮

    if (file.isInvalid()) {
        throw new FileNotFoundException("Invalid file path");
    }

    ⋮
```

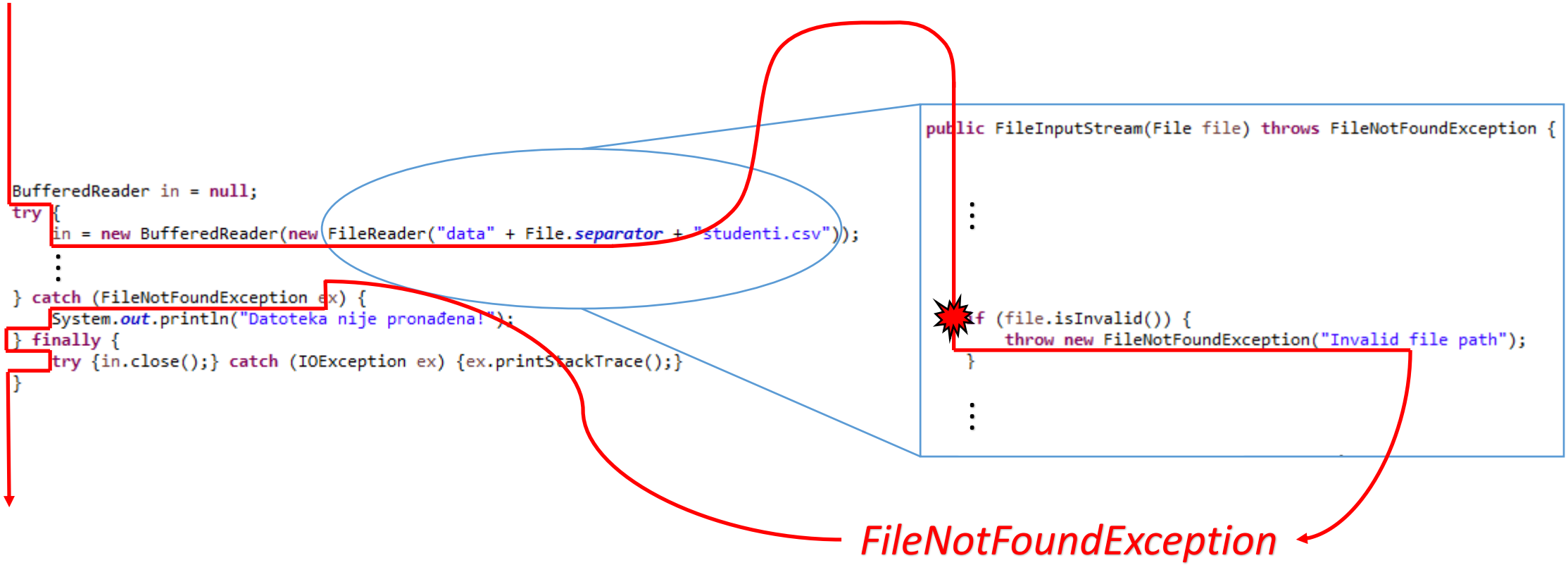
Izuzeci (*Exceptions*)

- uspešan tok:



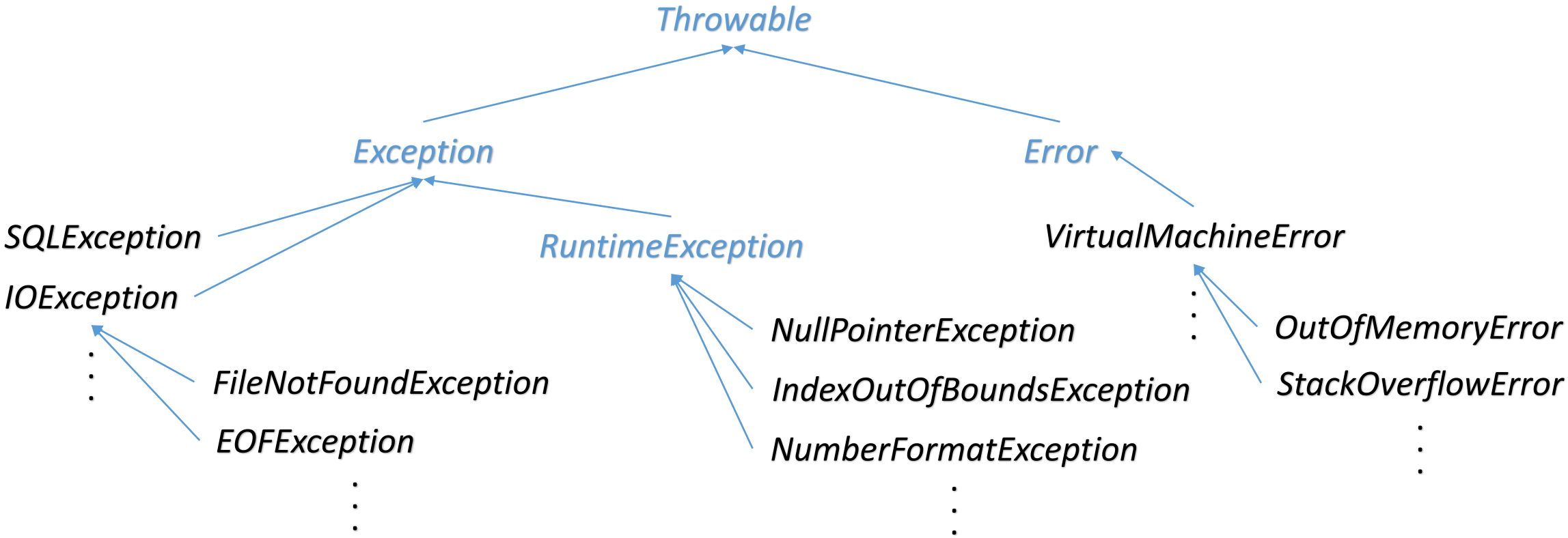
Izuzeci (*Exceptions*)

- neuspešan tok:



Izuzeci (*Exceptions*)

- izuzeci i greške su organizovani u proširivu hijerarhiju klasa koja omogućuje dodatnu fleksibilnost u korišćenju ovih mehanizama:



Izuzeci (*Exceptions*)

- bilo koji član hijerarhije izuzetaka može da se nasledi:
  - klasa koja nasleđuje *Throwable* može da stoji u *throw* naredbi i u *throws* i *catch* klauzulama
  - klasa koja nasleđuje *Exception* postaje *checked* izuzetak
  - klasa koja nasleđuje *RuntimeException* postaje *unchecked* izuzetak

- klasa naslednica izuzetka je *Java* klasa, pa može da poseduje atribute i metode

Izuzeci (*Exceptions*)

- *multi-catch* blok:

```
BufferedReader in = null;
try {
    in = new BufferedReader(new FileReader("data" + File.separator + "studenti.csv"));

    String linija = in.readLine();
    while (linija != null) {
        String[] tokeni = linija.split(",");

        Student student = new Student(tokeni[0], tokeni[1], tokeni[2]);
        studenti.add(student);

        linija = in.readLine();
    }
} catch (FileNotFoundException ex) {
    System.out.println("Datoteka nije pronađena!");
} catch (IOException ex) {
    System.out.println("Došlo je do greške u čitanju fajla!");
} catch (IndexOutOfBoundsException ex) {
    System.out.println("Neispravan sadržaj datoteke!");
} catch (Exception ex) {
    System.out.println("Došlo je do nepredviđene greške!");
} finally {
    try {in.close();} catch (Exception ex) {}
}
```

Izuzeci (*Exceptions*)

- ako je u *throws* klauzuli naveden izuzetak, on ne mora da se "uhvati" u funkciji, a može
- ako nešto u slučaju izuzetka ipak mora da se obradi u funkciji, to može da stoji u *try-finally* bloku:

```
public static List<Student> ucitaj() throws FileNotFoundException, IOException, IndexOutOfBoundsException {
    List<Student> studenti = new ArrayList<>();

    BufferedReader in = null;
    try {
        in = new BufferedReader(new FileReader("data" + File.separator + "studenti.csv"));

        String linija = in.readLine();
        while (linija != null) {
            String[] tokeni = linija.split(",");

            Student student = new Student(tokeni[0], tokeni[1], tokeni[2]);
            studenti.add(student);

            linija = in.readLine();
        }
    } finally {
        in.close();
    }

    return studenti;
}
```

Primer 2

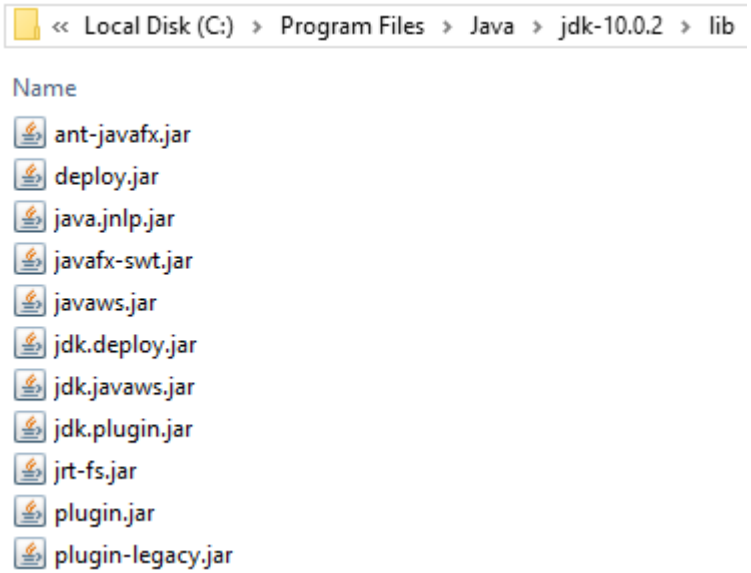
- A_DemoUncheckedException
- B_DemoCheckedException
- C_DemoHijerarhijaIzuzetaka
- D_DemoPropagacijaIzuzetaka
- E_TryWithResources
- F_DemoNasledjivanjeIzuzetaka
- G_DemoKontrolaToka

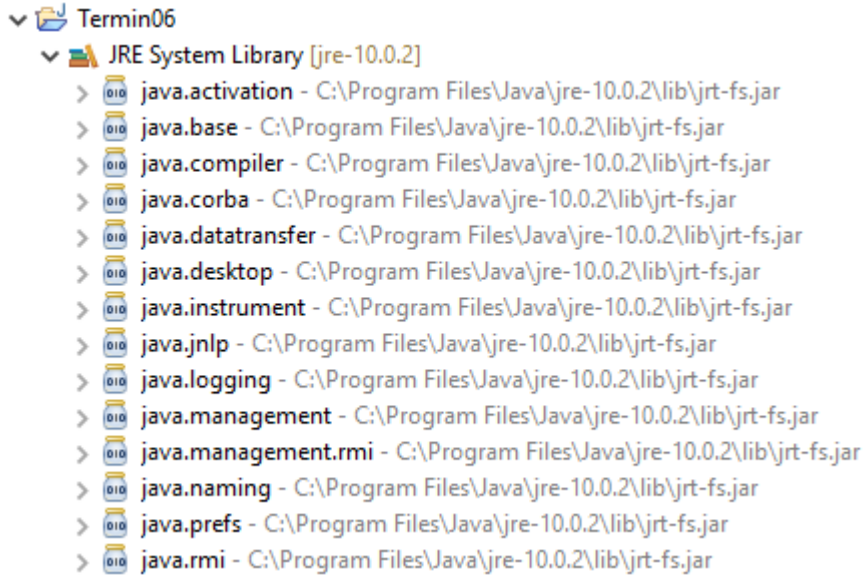
1. Osvrt na kolekcije
2. Mape
3. Iterator
4. Izuzeci
5. *Third-party* biblioteke

6. Kreiranje *.jar* datoteke
7. Dodatni materijali
8. Vežbanje

Third-party biblioteke

- većina *Java* klasa koje se koriste su instalirane uz JDK (*Java Development Kit*)
- klase su organizovane po *.jar* (*Java ARchive*) datotekama
- *.jar* datoteke moraju u sebi da sadrže prevedeni (*bin*) kod klasa, a mogu da sadrže i izvorni (*source*) kod i dokumentaciju (*doc*)
- razvojno okruženje prepoznaje ove datoteke i uvezuje ih u svaki projekat nakon čega ih prevodilac prepoznaje i mogu se *import*-ovati i koristiti





Third-party biblioteke

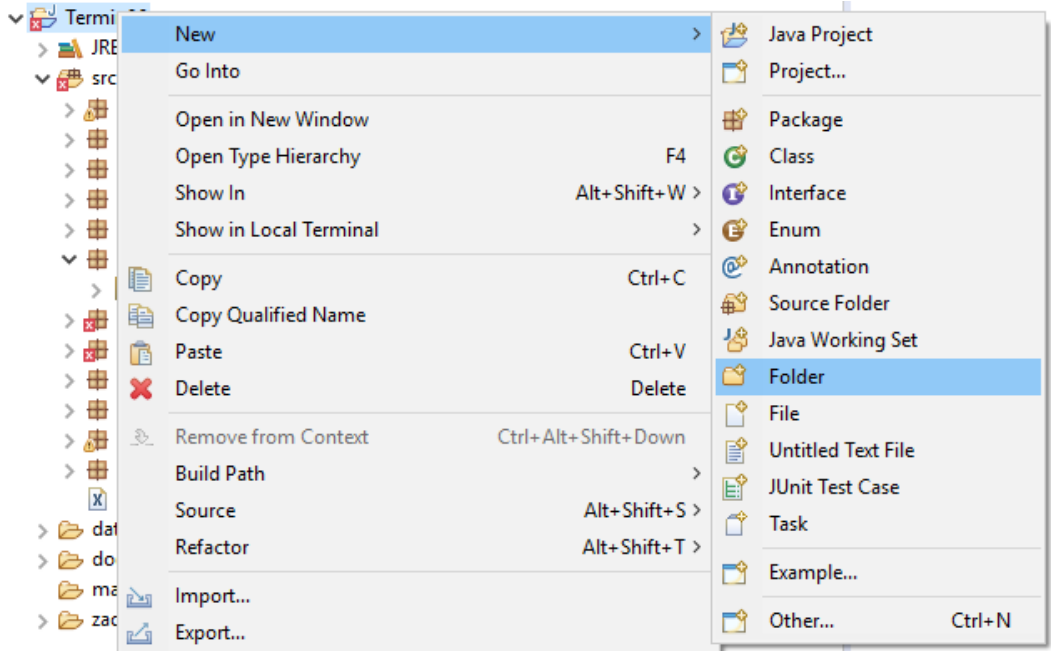
- skup standardnih *Java* klasa čine *Java Standard API* (*Application Programming Interface*)
- ukoliko one ne nude funkcionalnost koja nam je potrebna možemo:
  - sami da je implementiramo, ili
  - da sprečimo gubljenje vremena ako je neko drugi to već uradio

- *third-party* biblioteke predstavljaju organizovani skup klasa koje nude neku funkcionalnost ili skup funkcionalnosti, a nisu obuhvaćene *Java Standard API*-em, a koje je neko drugi napisao i dao na korišćenje
- sastoje se iz jedne ili više *.jar* datoteka sa pratećom dokumentacijom i opciono, izvornim kodom
- do njih se dolazi pretragom (*Google*), kroz literaturu, praćenjem sajtova za podršku programiranju i sl.

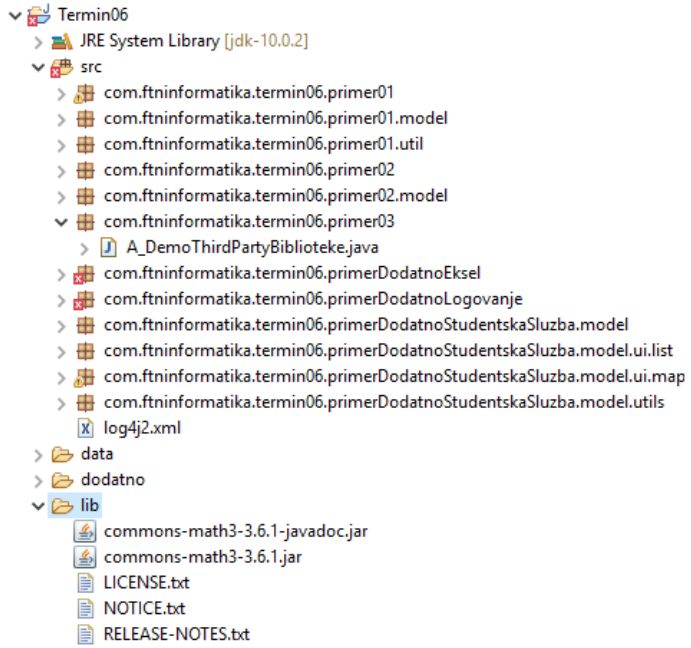
Third-party biblioteke

- za razliku od *Java Standard API*-a, prevodilac neće automatski prepoznati *third-party* biblioteke

1. potrebno je obezbediti njihovu kopiju u projektnom direktorijumu (bilo gde, ali se po konvenciji smeštaju u poddirektorijum nazvan *lib*)

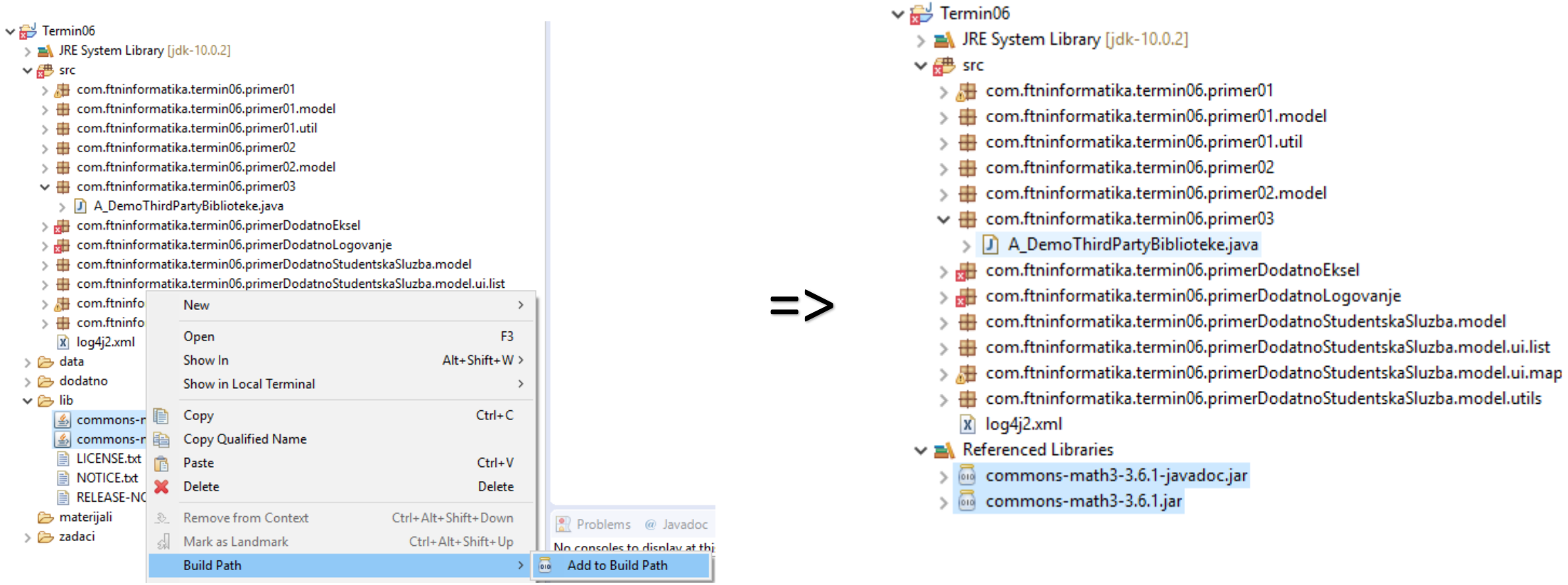


=>



Third-party biblioteke

2. zatim je potrebno uvezati ih u projekat (dodati ih u *build path*), da bi prevodilac mogao da ih prepozna



Primer 3

- A_DemoThirdPartyBiblioteke

1. Osvrt na kolekcije
2. Mape
3. Iterator
4. Izuzeci
5. *Third-party* biblioteke

6. Kreiranje *.jar* fajla
7. Dodatni materijali
8. Vežbanje

Kreiranje *.jar* datoteke

- postoje 2 razloga da sami kreiramo *.jar* datoteku:
  1. želimo da napravimo svoju *third-party* biblioteku i damo je na korišćenje

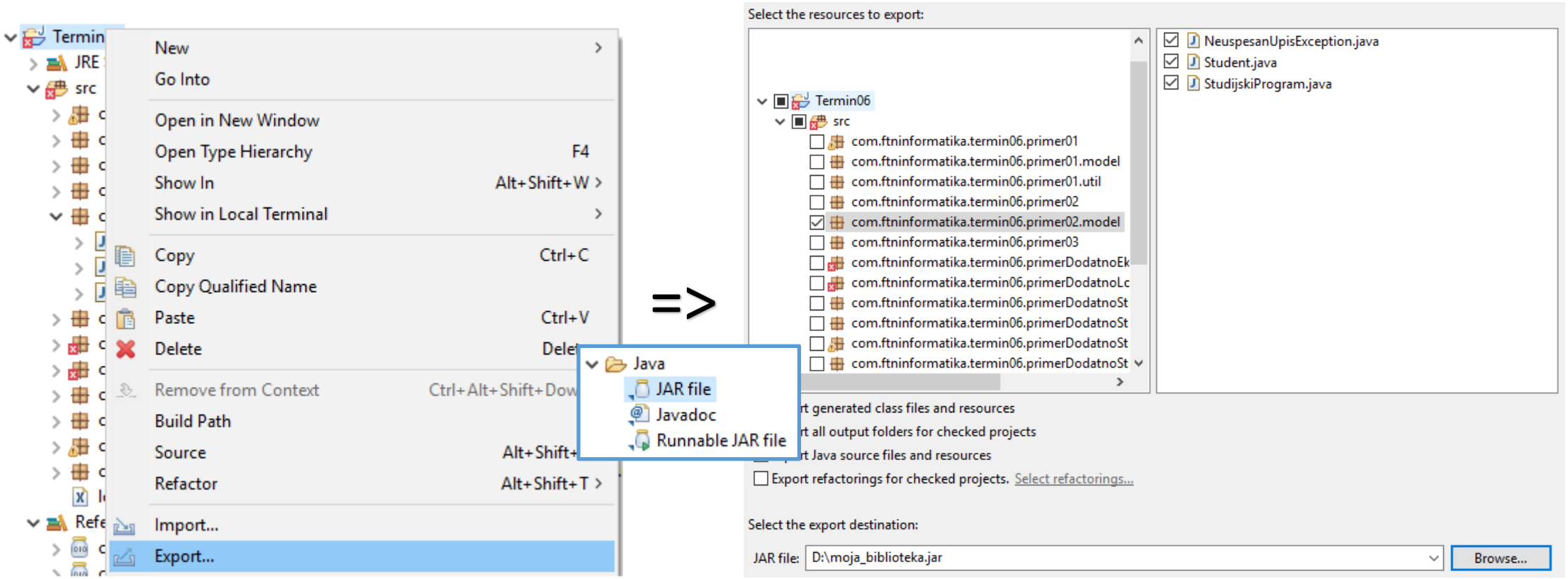
  u ovom slučaju potrebno je kreirati običnu *.jar* datoteku

  2. želimo da omogućimo da se naš *Java* program pokrene samostalno bez razvojnog okruženja i damo ga na korišćenje

  u ovom slučaju potrebo je kreirati izvršnu (*runnable*) *.jar* datoteku

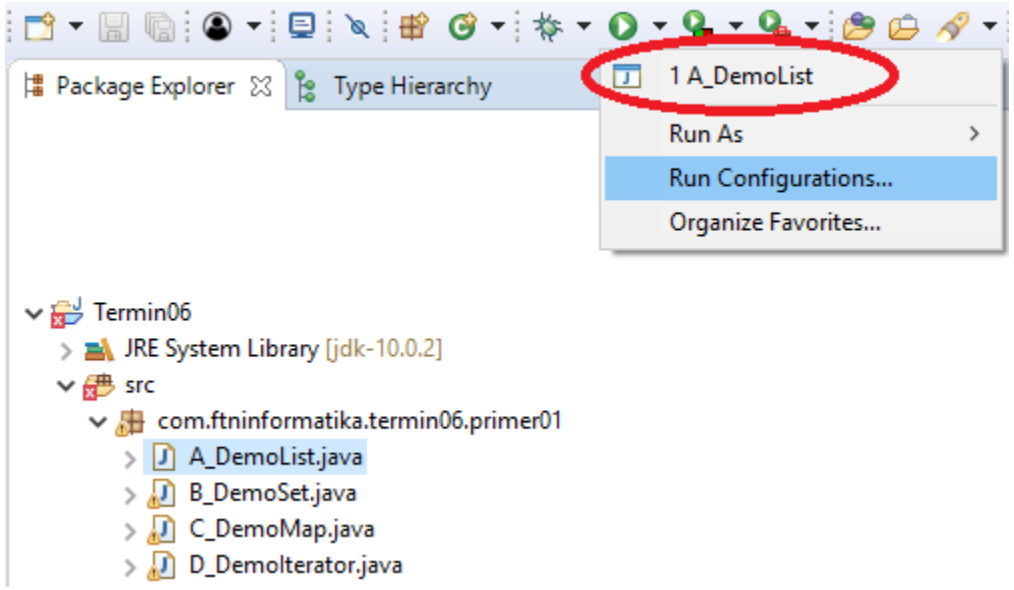
Kreiranje *.jar* datoteke

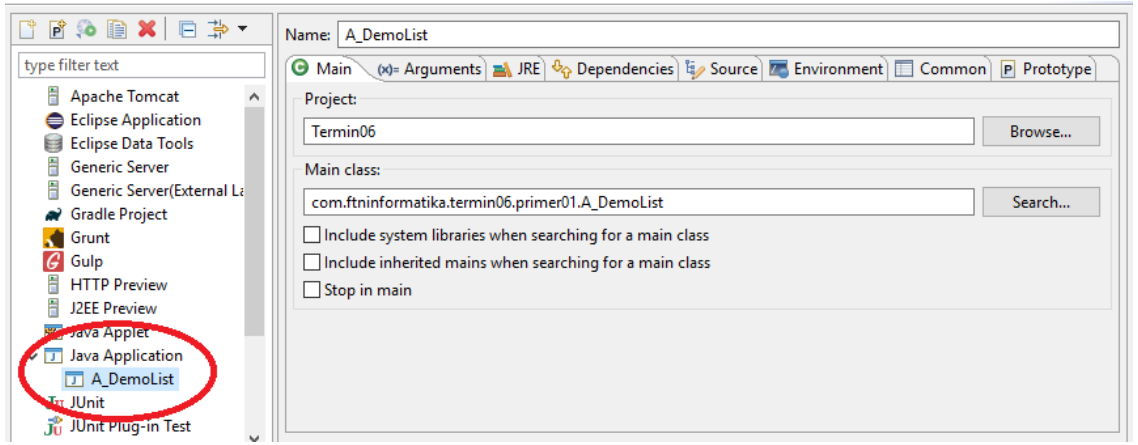
- primer kreiranja biblioteke *moja_biblioteka.jar* koja se sastoji od klasa iz paketa *primer02.model* (obična *.jar* datoteka):



Kreiranje *.jar* datoteke

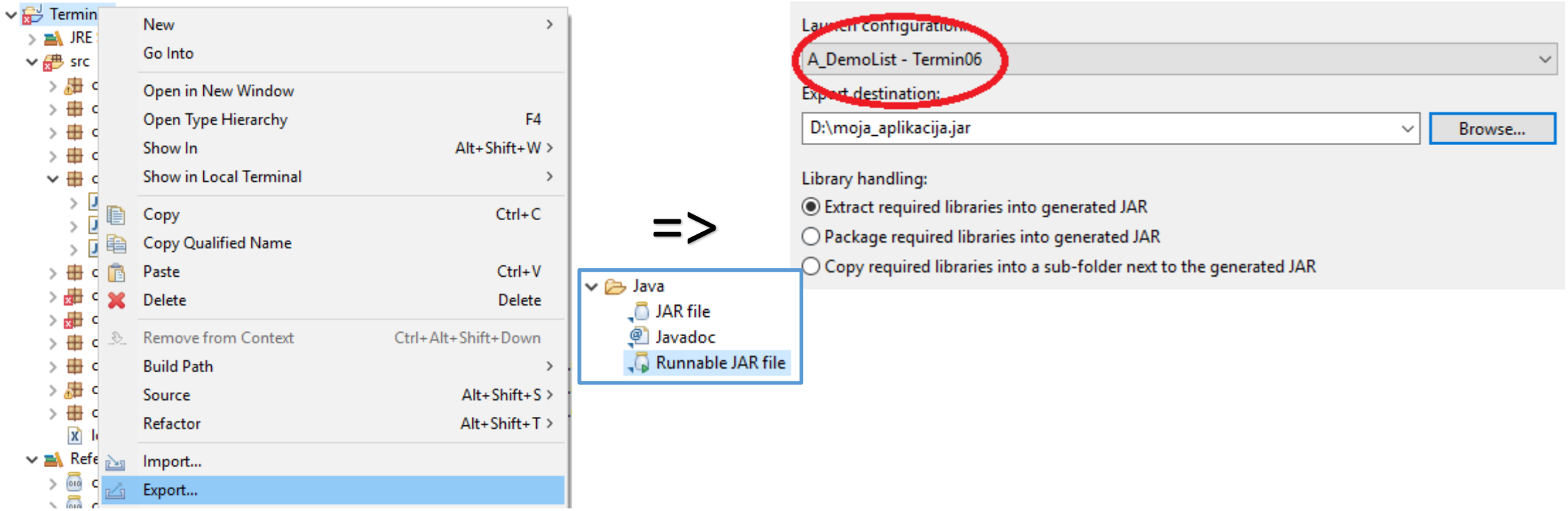
- projekat može imati više klasa sa *main* metodama (*Termin06* je jedan takav projekat jer svaki primer ima svoju *main* metodu)
- pri kreiranju izvršne *.jar* datoteke, neophodno je odabrati koja *main* metoda će započeti izvršavanje programa kada se *.jar* datoteka pokrene
- prvo pokretanje svake nove *main* metode u projektu ostavlja trag u vidu konfiguracije:





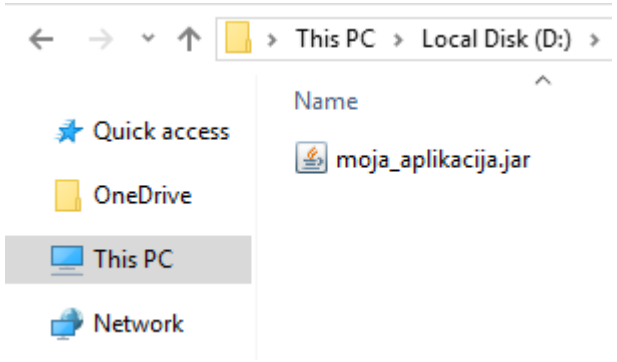
Kreiranje *.jar* datoteke

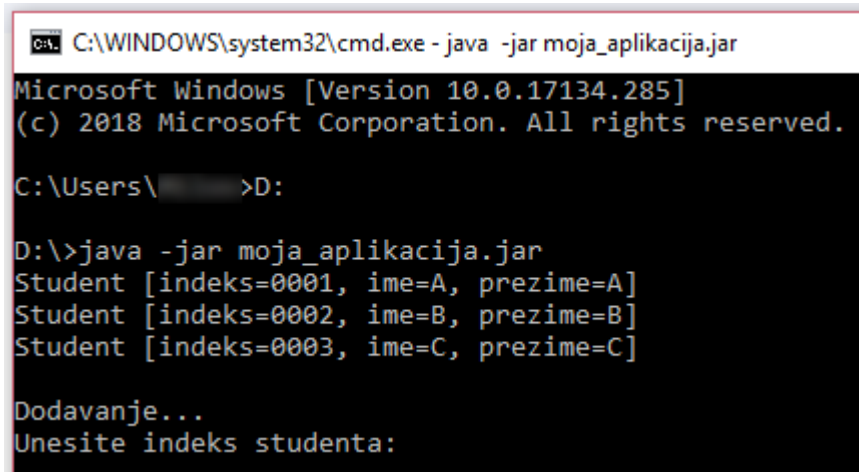
- pri kreiranju izvršne *.jar* datoteke, potrebno je odabrati konfiguraciju koja odgovara onoj *main* metodi koja će započeti izvršavanje programa kada se *.jar* datoteka pokrene
- primer kreiranja izvršne *.jar* datoteke *moja_aplikacija.jar* čijim se pokretanjem izvršava *main* metoda klase *A_DemoList.java*:



Kreiranje *.jar* datoteke

- pokretanje izvršne *.jar* datoteke se obavlja iz komandne linije komandom: *java –jar <naziv_datoteke>*:





1. Osvrt na kolekcije
2. Mape
3. Iterator
4. Izuzeci
5. *Third-party* biblioteke

6. Kreiranje *.jar* fajla
7. Dodatni materijali
8. Vežbanje

Dodatni materijali

1. implementacija dela Studentske službe uz pomoć liste i mape
2. primeri *third-party* biblioteka:
   - *Log4j* (biblioteka za vođenje dnevnika aplikacije)
   - *Apache POI* (biblioteka za rad sa *.xls* datotekama)

Vežbanje

- na času:
  1. *Zadatak za test - Termin 6.pdf*

- za domaći:
  1. završiti *Zadatak za test - Termin 5.pdf*
  2. završiti *Zadatak za test - Termin 6.pdf*
  3. dopuniti rešenje *Zadatak za test - Termin 6.pdf* da koristi *Apache POI* biblioteku za pisanje/čitanje podataka u/iz *.xls* umesto *.csv* datoteka
  4. *Zadatak - kolizije.pdf*