Praktikum Programski Sistemi

Upravljanje izuzecima u Javi

Postupanje sa greškama

- Ispravni i robusni programi
- Program je ispravan ako radi tačno ono što treba
- Program je robustan ako se u nepredviđenim okolnostima ponaša na razuman način

Postupanje sa greškama

- Primer: sortiranje brojeva
- Program mora uvek da bude ispravan
  - Program koji pogrešno sortira brojeve je beskoristan
  - Robustan program mora pravilno da postupa sa pogrešnim ulaznim podacima
- Program ne mora uvek da bude robustan
  - npr. pomoćni program koji pišemo za svoje potrebe ne mora uvek da bude robustan

Postupanje sa greškama

- Posledice loših programa mogu da budu banalne, ali i vrlo ozbiljne
  - Nerviranje zbog “kraha”, izgubljeno vreme i rad
  - Gubljenje novca i(li) ugleda
  - Ozbiljne posledice zbog loše programiranih medicinskih aparata, kosmičke opreme i sl.

Postupanje sa greškama u Javi

- Java ima nekoliko ugrađenih foolproof mehanizama za izbegavanje grešaka
  - sve promenljive su deklarisanog tipa
  - proveravaju se granice nizova (bounds checking)
  - direktno manipulisanje pokazivačima nije moguće
  - "curenje" memorije gotovo da nije moguće zbog ugrađenog sakupljača smeća (garbage collector)

Postupanje sa greškama

- Robustni program mora preživeti neobične i neočekivane okolnosti
- Jedan način za postupanje sa problemima jeste da se predvide svi mogući problemi i da se u kôd uključi testiranje na njih

Postupanje sa greškama

- Primer: korišćenje elemenata niza a

```
if (i < 0 || i > a.length)
{
  // postupanje sa problemom
}
else
{
  // normalna obrada elementa a[i]
}
```

Nedostaci pristupa testiranja

- Teško je, a najčešće i nemoguće predvideti sve moguće probleme
- Nije uvek jasno šta treba preduzeti kada se otkrije problem
- Čak i jednostavan program postaje mešavina pravih naredbi i “if” naredbi

Izuzeci (exceptions)

- Drugi način za postupanje sa problemima u Javi su **izuzeci** (exceptions)
- Izuzetak je opštiji pojam od greške, jer obuhvata sve okolnosti koje predstavljaju odstupanje od predviđenog toka programa
  - sve greške
  - specijalni slučajevi na koje programer nema uticaja ili narušavaju normalan tok programa

Izuzeci

- Kada se tokom izvršavanja desi izuzetak, kaže se da je on bačen (thrown)
- Ako se izuzetak ne obradi, izvršavanje programa se prekida
- Izuzetak mora da se obradi, ili "uhvati" (catch)

Izuzeci

- U Javu je ugrađen sistem za upravljanje izuzecima, tj. hijerarhija klasa izuzetaka je deo jezika
- Kada se desi izuzetak, baca se objekat klase izuzetka koji sadrži:
  - poruku o tome šta je dovelo do izuzetka
  - listu metoda koji su se izvršavali kada se izuzetak dogodio (method call stack, stack trace)

Klasa Throwable

- Svi objekti izuzetka moraju pripadati nekoj klasi izvedenoj iz osnovne klase izuzetaka **java.lang.Throwable**
- Klasa **Throwable** sadrži nekoliko metoda koji se mogu koristiti sa svakim objektom izuzetka
  Throwable e;

Klasa Throwable

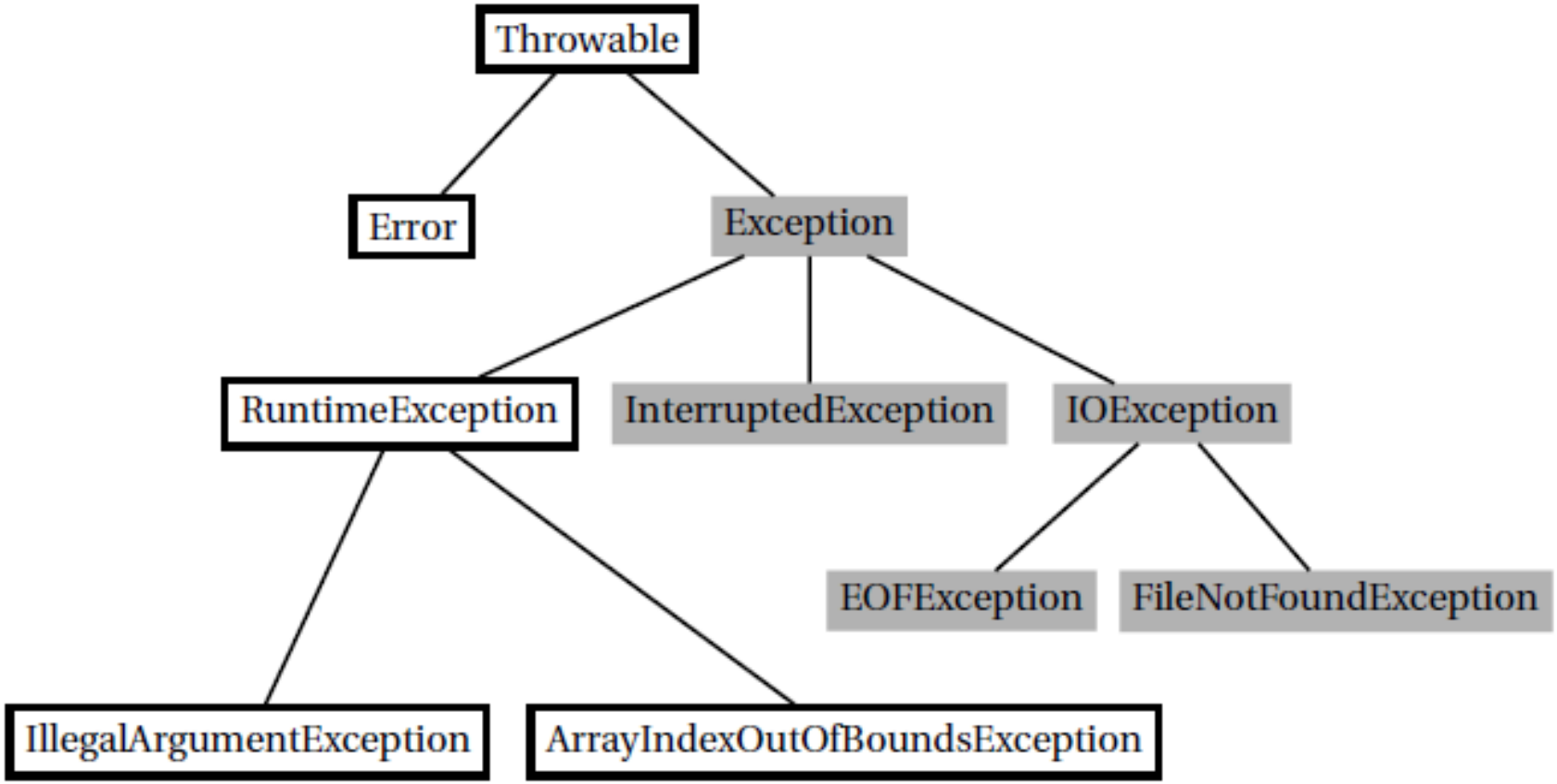
```
Throwable e;
```

`e.getMessage()` – poruka o grešci

`e.toString()` – stringovna reprezentacija objekta izuzetka

`e.printStackTrace()` – lista aktivnih metoda (Method Call Stack)

Hijerarhija klasa izuzetaka



Deo hijerarhije klasa izuzetaka

Hijerarhija izuzetaka

- Klasa **Throwable** ima dve direktne klase naslednice, **Error** i **Exception**
  - Ove dve klase imaju svoje unapred definisane mnoge druge klase naslednice
- Ako postojeće klase izuzetaka nisu odgovarajuće za konkretnu primenu, u programu se mogu definisati nove klase izuzetaka radi predstavljanja posebnih tipova grešaka

Klasa Error

- Klasa **Error** i njene naslednice predstavljaju vrlo ozbiljne greške unutar samog Java interpretatora
  - obično se radi o fatalnim greškama koje izazivaju prekid izvršavanja, pošto ne postoji razuman način da se program oporavi
  - Primer: neispravan bajtkod daje izuzetak **ClassFormatError**

Klasa Exception

- Izuzeci tipa (ili podtipa od) **Exception** su predviđeni da budu uhvaćeni i obrađeni u programu na odgovarajući način
- Među naslednicama klase **Exception** je klasa **RuntimeException** koja opisuje najčešće greške koje se mogu desiti tokom izvršavanja programa

Neproveravani izuzeci

- Izuzeci tipa **Error** i **RuntimeException** su tzv. neproveravani izuzeci (unchecked exceptions)
  - to znači da njihovo hvatanje i obrada u programu nisu obavezni, već se programeru ostavlja izbor da li je bolje u programu reagovati na ove izuzetke ili prosto dozvoliti da se izvršavanje programa nasilno prekine

Proveravani izuzeci

- Klasa **Exception** i njene naslednice (osim klasa **RuntimeException**) čine grupu proveravanih izuzetaka (checked exceptions)
  - Tim izuzecima mora se rukovati u programu
  - Ako u programu postoji mogućnost njihovog izbacivanja, oni se u programu moraju hvatati i obraditi

Hvatanje i obrada izuzetaka

```
try
{
  // naredbe koje mogu dovesti do izuzetka
}
catch (Exception e)
{
  // naredbe koje obrađuju izuzetak
}
```

Izvršavanje try-catch bloka

1. Izvršavaju se naredbe u bloku **try**
2. Ako se ne baci nijedan izuzetak, ili se izbaci izuzetak različitog tipa od onog navedenog u zagradama iza **catch**, blok **catch** se preskače
3. Ako se izbaci izuzetak tipa navedenog u zagradama iza **catch**, izvršava se blok **catch**
4. Izvršavanje se nastavlja iza bloka **catch**

Primer

```
try {
   a[i] = 0;
}
catch (ArrayIndexOutOfBoundsException e) {
   System.out.println("Indeks izvan granica");
   e.printStackTrace();
}
```

Primer

```
try {
   a[i] = 0;
}
catch (ArrayIndexOutOfBoundsException e) {
   System.out.println("Indeks izvan granica");
   e.printStackTrace();
}
catch (NullPointerException e) {
   System.out.println("Niz a ne postoji");
   e.printStackTrace();
}
```

Primer

- Klase **ArrayIndexOutOfBoundsException** i **NullPointerException** su naslednice klase **RuntimeException**, pa se prethodni primer može kraće zapisati:

```
try {
  a[i] = 0;
}
catch (RuntimeException e) {
  System.out.println("Greška u radu sa nizom a");
  System.out.println("Greška je " + e);
  e.printStackTrace();
}
```

Primer

- Prethodni primer ilustruje zašto su klase izuzetaka organizovane u hijerarhiju
- U posebnim slučajevima hvataju se specifični tipovi izuzetaka, ali i mnogo širi tipovi izuzetaka ukoliko nije bitan njihov poseban tip
  - izuzetak tipa **NullPointerException** bi se uhvatio klauzulama **catch** za tipove **NullPointerException**, **RuntimeException**, **Exception** ili **Throwable**.

Klauzula finally

- Naredba **try** u Javi može imati dodatnu klauzulu **finally**
- Koristi se u slučajevima kada je nakon izuzetka potrebno vratiti program u konzistentno stanje pre pojave greške
  - primer: zatvaranje fajla, oslobađanje resursa i sl.

Klauzula finally

```
try {
    .
    . // Naredbe koje mogu izazvati izbacivanje izuzetaka
    .
}
catch ( ... ) {
    .
    . // Naredbe koje obrađuju izuzetak određenog tipa
    .
}
.
. // Ostali catch blokovi
.
finally {
    .
    . // Naredbe koje se uvek izvršavaju
    .
}
```

Klauzula finally

- Blok **finally** se uvek izvršava kao poslednji korak izvršavanja bloka **try**, bez obzira da li je izuzetak bačen i bez obzira da li je bačeni izuzetak uhvaćen i obrađen
- Blok **finally** je namenjen za postupak "čišćenja za sobom" koji se nikako ne sme preskočiti

Izbacivanje izuzetka u programu

- Izuzetak se može izbaciti programski
  - Postoje situacije kada program otkriva izuzetak, ali se on na tom mestu ne može razumno obraditi
  - Izuzetak može biti bačen sa ciljem da on bude obrađen na nekom drugom mestu u programu
- Za bacanje izuzetka služi ključna reč **throw**

  throw objekatIzuzetka;

Bacanje izuzetka

- U naredbi ***throw*** *objekatIzuzetka;* *objekatIzuzetka* mora biti objekat klase koja nasleđuje **Throwable** (ili češće, nekoj klasi izvedenoj iz **Exception**)
  - Primer:
    throw new ArithmeticException(“Deljenje nulom”);

Bacanje izuzetka

- Da bi se naznačilo da u telu nekog metoda može doći do bacanja izuzetka, zaglavlju metoda se može dodati klauzula **throws**
- Primer:

```
public void nekiMetod( ... ) throws ArithmeticException {
    . . .
}
```

- Ako u metodu može doći do bacanja izuzetka više tipova, ovi tipovi se navode iza klauzule **throws** razdvojeni zarezima

Postupanje sa izuzecima

```
void A( ... ) {
    . . .
    B( ... )  // poziv metoda B
    . . .
}

void B( ... ) {
    . . .
    ⚡🛑⚡     // naredba koja je uzrok izuzetka
    . . .
}
```

Postupanje sa izuzecima

- Dva moguća slučaja
  1. Izuzetak uhvaćen u metodi B
     - Izvršava se odgovarajući catch blok
     - Izvršava se klauzula finally, ako postoji
     - Nastavlja se normalno izvršavanje ostatka metoda B
  2. Izuzetak nije uhvaćen u metodi B
     - sa izuzetkom se postupa na isti način kao da je izbačen izvan nekog try-catch bloka
     - metoda A dobija šansu da obradi bačen izuzetak

Postupanje sa izuzecima

- Lanac poziva metoda se odmotava
- Postupanje sa bačenim izuzetkom se nastavlja sve dok se on ne obradi, ili dok se ne stigne do metode main

Proveravani izuzeci

- Izuzeci klasa izvedenih iz **Exception** (sa izuzetkom **RuntimeException**) moraju se obraditi na jedan od dva načina:
  - Stavljanjem naredbe koja može da baci neproveravani izuzetak u odgovarajući try-catch blok
  - Navođenjem klauzule **throws** u zaglavlju metoda koji sadrži naredbu koja može da baci neproveravani izuzetak

Proveravani izuzeci

- U prvom slučaju, proveravani izuzetak biće uhvaćen u metodu u kojem je nastao, pa nijedan drugi metod koji poziva izvorni metod neće ni znati da se desio izuzetak
- U drugom slučaju, proveravani izuzetak neće biti uhvaćen u metodu u kojem je nastao, pa svaki metod koji poziva izvorni metod mora da obradi originalni izuzetak.

Prvi slučaj

```
void čitajDatoteku (String imeDatoteke) {

   FileInputStream datoteka;
   . . .
   try {
      datoteka = new FileInputStream(imeDatoteke);
   }
   catch (FileNotFoundException e) {
      System.out.println("Datoteka " + imeDatoteke + " ne postoji");
      return;
   }
   // Naredbe za čitanje datoteke
   . . .
}
```

Drugi slučaj

```
void čitajDatoteku (String imeDatoteke)
    throws FileNotFoundException {

    FileInputStream datoteka;
    . . .
    datoteka = new FileInputStream(imeDatoteke);
    // Naredbe za čitanje datoteke
    . . .
}
```

Definisanje klasa izuzetaka

- Ako nijedna standardna klasa nije odgovarajuća za neku potencijalnu grešku, programeri mogu definisati svoje klase izuzetaka
- Nova klasa izuzetaka mora biti naslednica klase **Throwable** ili neke njene naslednice
  - Obično je naslednica klase **Exception** jer želimo da bude proveravan izuzetak

Primer

```
public class PogrešanPrečnikIzuzetak extends Exception {

    // Konstruisanje objekta izuzetka
    // koji sadrži datu poruku o grešci
    public PogrešanPrečnikIzuzetak(String poruka) {
      super(poruka);
    }
}
```

Tri zlatna pravila za izuzetke

- Budite što određeniji kod tipova izuzetaka
  - Potrebno je uložiti dodatni napor za kodiranje nekoliko catch blokova, ali se to uvek na duži rok isplati
- Bacajte izuzetak gde god se ukaže prilika
- Hvatajte izuzetke tek na mestu gde znate šta ćete sa njima da radite

Pravilo br. 1

```
File prefsFile = new File(prefsFilename);

try
{
    readPreferences(prefsFile);
}
catch (FileNotFoundException e)
{
    // alert the user that the specified file
    // does not exist
}
catch (EOFException e)
{
    // alert the user that the end of the file
    // was reached
}
catch (ObjectStreamException e)
{
    // alert the user that the file is corrupted
}
catch (IOException e)
{
    // alert the user that some other I/O
    // error occurred
}
```

Pravilo br. 2

```
public void readPreferences(String filename)
    throws IllegalArgumentException
{
    if (filename == null)
    {
        throw new IllegalArgumentException
                                ("filename is null");
    }  //if

    //...perform other operations...

    InputStream in = new FileInputStream(filename);

    //...read the preferences file...
}
```

Pravilo br. 3

```
public void readPreferences(String filename)
{
    //...

    InputStream in = null;

    // DO NOT DO THIS!!!
    try
    {
        in = new FileInputStream(filename);
    }
    catch (FileNotFoundException e)
    {
        logger.log(e);
    }

    in.read(...);

    //...
}
```

```
public void readPreferences(String filename)
    throws IllegalArgumentException,
           FileNotFoundException, IOException
{
    if (filename == null)
    {
        throw new IllegalArgumentException
                    ("filename is null");
    }  //if

    //...

    InputStream in = new FileInputStream(filename);

    //...
}
```