10

Iznimke

- Što je iznimka?
- Hvatanje iznimki u Javi
- Procesiranje iznimki

Što je iznimka?

Napisali smo program provjerili ga za neke svoje podatke i sve radi kako treba, testiramo program ponovo, kad gle, sada ne radi, izbacuje grešku. U čemu je problem? Razloga zašto je program maloprije radio a sada više ne radi je puno. Neki od češćih su:

- program čita podatke iz datoteke, koju smo u međuvremenu izbrisali ili preimenovali;
- umjesto broja smo unijeli znak;
- podaci su takvi da se u nekom trenutku pojavljuje dijeljenje s nulom;
- ...

Sve gore navedene situacije u stvari nisu greške, već specifične situacije za koje naš program ne radi ispravno tzv. **iznimke**. Kako takvih iznimnih situacija može biti jako puno i teško ih je sve predvidjeti, Java nudi mogućnost automatskog praćenja iznimaka.
Jednostavno ćemo u kodu doati do znanja da je neki dio programa problematičan i da bi se u njemu mogla pojaviti iznimka. Ukoliko se iznimka ne pojavi, nikome ništa, program će se izvršiti kako smo i htjeli i to je u stvari očekivano ponašanje. Međutim, ako se iznimka pojavi, program neće stati s radom, kao inače, već ćmo mi reći što treba napraviti u takvoj situaciji. Najčešće će to biti poruka korisniku da se je pojavila greška, pokušaj ispravljanja greške predviđanjem onoga što je korisnik programa htio napraviti,...

Hvatanje iznimki u Javi

U programu možemo specificirati da pratimo samo određenu vrstu iznimki, npr. pratimo samo iznimke koje mogu uslijediti zbog unosa pogrešnog tipa podatka, iznimke vezane uz pristup fileovima, iznimke vezane uz formate brojeva,...

Dio koda koji bi mogao prouzročiti iznimku u programu stavljamo u blok **`try`**. Nakon **`try`** bloka slijedi **`catch`** blok, kojih može biti više. U bloku **`catch`** specificiramo vrstu iznimke koju pratimo. Iza posljednjeg **`catch`** bloka opcionalno dolazi blok **`finally`**. Unutar bloka **`finally`** stavljamo dio programa koji se izvršava uvijek, bez obzira je li došlo do greške ili nije. **`Finally`** blok je opcionalan, međutim ako nismo stavili niti jedan **`catch`** blok, nakon **`try`** moramo staviti **`finally`** blok.

```
try
{
      //naredbe koje bi mogle prouzrokovati iznimke
}
catch (VrstaIznimke1 iz1)
{
      /*
      dio koda koji se izvršava ako se je pojavila iznimka tipa
      VrstaIznimke1
      */
}
...
finally
{
      /*
      dio programa koji se uvijek izvršava, bez obzira je li došlo do
      iznimke ili ne
      */
}
```

Ukoliko se u nekoj od naredbi unutar bloka **try** pojavi iznimka prelazi se na prvi **catch** blok, ukoliko taj **catch** blok "nadležan" za praćenje iznimke koja se je pojavila u bloku **try** izvršava se kod unutar tog **catch** bloka, inače se prijelazi na sljedeći **catch** blok,... Na kraju se uvijek (bez obzira je lise iznimka pojavila ili ne) prelazi na **finally** blok, ukoliko on postoji. Ukoliko iza bloka **try** ne postoji blok **catch**, tada se odmah prelazi na blok **finally**, koji u tom slučaju morapostojati.

Već smo rekli da postoji nekoliko vrsta iznimki koje možemo pratiti i tada naziv te iznimke navodimo kod **catch** bloka. Vrste iznimki definirane su posebnim klasama. Osnovna klasa koja nam omogućava praćenje svih iznimaka u programu je klasa **Exception**. Sve ostale klase za praćenje iznimki nasljeđuju klasu **Exception**. Kao i svaka druga klasa, **Exception**, pa onda i sve druge klase za praćenje iznimaka imaju svoje metode. Najčešće korištene metode su:

Metoda	Opis
getMessage ()	**vraća opis iznimke**
toString ()	**vraća vrstu iznimke, odnosno naziv klase koja je stvorila iznimku**
printStackTrace ()	**koristi se kod nasljeđivanja klasa a koristi se kada se želi pratiti u kojim su se klasama i metodama greške pojavljivale**

Tablica 10 – 1 Metode klase **Exception**

Kao što smo već rekli, osim genralne klase **Exception**, koja prati sve iznimke koji se mogu pojaviti, postoji i niz drugih klasa za praćenje određenog tipa iznimki, a koje nasljeđuju klasu **Exception**.
Neke od klasa za praćenje iznimaka dane su u sljedećoj tablici:

Klasa iznimaka	Opis
ArithmeticException	**Aritmetičke greške kao dijeljenje s 0,...**
FileNotFoundException	**Rad s fileom koji ne postoji**
StringIndexOutOfBoundsException	**Indeks u stringu ne postoji**
NumberFormatException	**Nedozvoljeni format broja**

Tablica 10 – 2 Najčešće klase za praćenje iznimaka

Korištenje klasa za automatsko praćenje iznimaka uvelike olakšava posao programera. U programima je bitno voditi računa da pratite pravu vrstu iznimki, kako neke vrste iznimki ne bi prošle neprimjećene, što bi u tom slučaju prouzročilo prekid u radu programa. Hvatanje apsolutno svih iznimaka korištenjem klase Exception je moguće ali nije baš preporučljivo.

Primjer 10 – 1:
Napišimo program koji će na ekranu nacrtati prozor. Prozor treba imati dva okvira za tekst, jedan ispod drugoga, ispod njih treba imati gumb na kojem piše *Zbroji* te ispod gumba još jedan okvir za tekst. Klikom na gumb *Zbroji* u treći okvir za tekst upiše zbroj brojeva koji su upisani u prva dva okvira za tekst. U slučaju da nešto od unesenog nije broj, u treći okvir za tekst treba ispisati odgovarajuću poruku.

Rješenje:
```
import javax.swing.*;
import java.awt.*;
```

```
import java.awt.event.*;
public class Zbrajanje extends JFrame implements ActionListener
{
    private final int sirina = 220;
    private final int visina = 200;
    private Container c;
    private JTextField t1, t2, t3;
    private JButton b;

    public Zbrajanje()
    {
        setTitle ("Zbrajanje");
        setSize (sirina, visina);
        setDefaultCloseOperation (EXIT_ON_CLOSE);

        c = getContentPane ();
        c.setLayout (null);

        t1 = new JTextField ();
        t1.setSize (200, 25);
        t1.setLocation (5, 5);
        c.add (t1);

        t2 = new JTextField ();
        t2.setSize (200, 25);
        t2.setLocation (5, 45);
        c.add (t2);

        b = new JButton ();
        b.setText ("Zbroji");
        b.setSize (200, 25);
        b.setLocation (5, 85);
        b.addActionListener (this);
        c.add (b);

        t3 = new JTextField ();
        t3.setSize (200, 25);
        t3.setLocation (5, 125);
        c.add (t3);

        setVisible (true);
    }

    public void zbroji ()
    {
        try
        {
            int a = Integer.parseInt (t1.getText ());
            int b = Integer.parseInt (t2.getText ());
            int c = a + b;
            t3.setText ("" + c);
        }
        catch (NumberFormatException e)
        {
            t3.setText ("Pogrešan format broja");
        }
    }

    public void actionPerformed (ActionEvent e)
    {
        zbroji ();
```

```
    }

    public static void main (String[] s)
    {
        Zbrajanje z = new Zbrajanje ();
    }
}
```

Procesiranje iznimki

Kako se u stvari prate iznimke u programu? Kada se unutar **try** bloka neke metode dogodi iznimka, neki dio metode se ne može korektno izvršiti, kontrola programa se preusmjeri na **catch** blok.

Promotrimo to na prethodnom primjeru. Metoda **zbroji ()** imala je sljedeći oblik:

```
public void zbroji ()
{
      try
      {
            int a = Integer.parseInt (t1.getText ());
            int b = Integer.parseInt (t2.getText ());
            int c = a + b;
            t3.setText ("" + c);
      }
      catch (NumberFormatException e)
      {
            t3.setText ("Pogrešan format broja");
      }
    }
```

Ukoliko korisnik u okvir za tekst *t1* unese vrijednost koja nije cijeli broj. Naredba: **`Integer.parseInt (t1.getText ());`** neće se moći uspješno izvršiti. te će se preskočiti sve druge naredbe unutar **`try`** bloka i izvršavanje programa će se nastaviti u **`catch`** bloku, tj. izvršit će se naredba: `t3.setText ("Pogrešan format broja");`
Što je u stvari u pozadini toga. Priča ide otprilike ovako. Programer koji je programirao metodu **`parseInt ()`** klase **`Integer`**, pretpostavio je da bi se često moglo dogoditi da parametar ove metode ne bude string koji se sastoji samo od znamenaka. U tom će se slučaju dogoditi nekoliko stvari:

- kreirati će se objekt iz neke od klasa za rad s iznimkama (u ovom slučaju je to specijalno klasa **`NubmerFormatException`**);
- izvršavanje metode (u našem slučaju `zbroji ()`) na tom mjestu prestaje te se traži mjesto za nastavak izvršavanja. To mjesto je blok **`catch`**.

Postavlja se pitanje, a možemo li mi, poput programera koji je programirao metodu **`parseInt ()`**, kod svojih metoda na ovakav način preusmjeravati izvršavanje programa i vraćati poruke o greškama? Naravno da možemo i to na sljedeći način:
Pretpostavimo da imamo metodu **`inicijali ()`**, čiji je parametar tipa string, koji predstavlja ime i prezime neke osobe, odvojene jednim razmakom, a metoda vraća inicijale te osobe (prvo slovo imena i prvo slovo prezimena)
Metoda **`inicijali ()`**, unutar svoje klase bi mogla izgledati ovako:

```
public class Inicijali
```

```
{

public static String inicijali (String ime)
{
      int n = ime.indexOf (" ");
      return ime.substring (0, 1) + ". " + ime.substring (n + 1, n + 2) +
".";
}
}
```

Budući da je metoda **`inicjali ()` static**, može se pozivati bez da se kreira objekt klase **`Inicijali`**, pa stoga klasa **`Inicijali`** ne mora imati konstruktor.

Napišimo sada odgovarajući program i testirajmo ovu metodu:

Primjer 10 – 2:
Napišimo program koji će crtati korisničko sučelje kao na slici. Klikom na gum *Inicijali*, u drugi okvir za tekst trebaju se ispisati inicijali osobe čije je ime i prezime uneseno u prvi tekstualni okvir.

Rješenje:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TestInicijali extends JFrame implements ActionListener
{
    private Container c;
    private JButton b1;
    private JTextField t1, t2;
    public TestInicijali()
    {
        setTitle ("Inicijali");
        setSize (300, 120);
        setDefaultCloseOperation (EXIT_ON_CLOSE);

        c = getContentPane ();
        c.setLayout (new GridLayout(3, 1));

        t1 = new JTextField ();
        c.add (t1);

        b1 = new JButton ();
        b1.setText ("Inicijali");
        b1.addActionListener (this);
        c.add (b1);

        t2 = new JTextField ();
        c.add (t2);
```

```
        setVisible (true);
    }

    public void actionPerformed (ActionEvent e)
    {
            t2.setText (Inicijali.inicijali (t1.getText ()));
    }

    public static void main (String[] s)
    {
        TestInicijali ti = new TestInicijali ();
    }
}
```

Pokrenemo li program i upišemo npr. u prvi okvir za tekst *Ivan Marković*, program će raditi dobro, no međutim upišemo li samo *Ivan*, program neće raditi dobro, a ako upišemo *Ivan* i na kraju stavimo još jedan razmak, potpuno će se srušiti. To rušenje programa nam se ne sviđa. Greška je u tome što se mi u svojom metodi **`inicijali ()`** na jednom mjestu referenciramo na prvi znak iza razmaka, no što ako razmak uopće ne postoji? Što ako nema znaka nakon prvog razmaka? To su izuzetne situacije i u njima bi program trebao javiti grešku, pa ispravimo svoju metodu **`inicijali ()`** tako da radi korektno:

```
public class Inicijali
{
public static String inicijali (String ime) throws Exception
{
      int n = ime.indexOf (" ");
      if (n == -1 || n == ime.length () - 1)
            throw new Exception ("Podaci nisu potpuni");
      else
            return ime.substring (0, 1) + ". " + ime.substring (n + 1, n +
2) + ".";
}
}
```

Kao što možemo primijetiti dodali smo u metodi provjeravanje uvjeta. Ukoliko nema razmaka ili je jedini razmak na posljendjem mjestu, izvršavanje metode koja poziva metodu **`inicijali ()`** će se prekinuti, kreirat će se objekt **`Exception`** te će se izvršavanje metode nastaviti u bloku **`catch`**. String koji prosljeđujemo konstruktoru klase **`Exception`** u stvari će biti tekst poruke koju će objekt tipa exception vratiti metodom **`getMessage ()`**, odnosno tekst prvog reda koji vraća metoda **`printStackTrace ()`**.
Primijetimo da smo izmijenili i zaglavlje metode **`inicijali ()`**, u nastavku smo dodali **`throws Exception`**, što u osnovi znači da metoda sve iznimke prosljeđuje klasi **`Exception`**. Umjesto **`Exception`** mogla se je ovdje naći i neka druga klasa nakoju bi prosljeđivali izvršavanje programa u slučaju greške.
Nakon što smo ovako izmijenili metodu inicijali, moramo izmijeniti i metodu actionPerformed () koja poziva metodu **`inicijali ()`**. Naime, u metodi **`actionPerformed ()`** dio koda u kojem se nalazi poziv metode inicijali trebamo staviti u **`try`** blok.
Tako izmijenjena metoda **`actionPerformed ()`** sada ima oblik:

```
public void actionPerformed (ActionEvent e)
{
      try
      {
            t2.setText (Inicijali.inicijali (t1.getText ()));
```

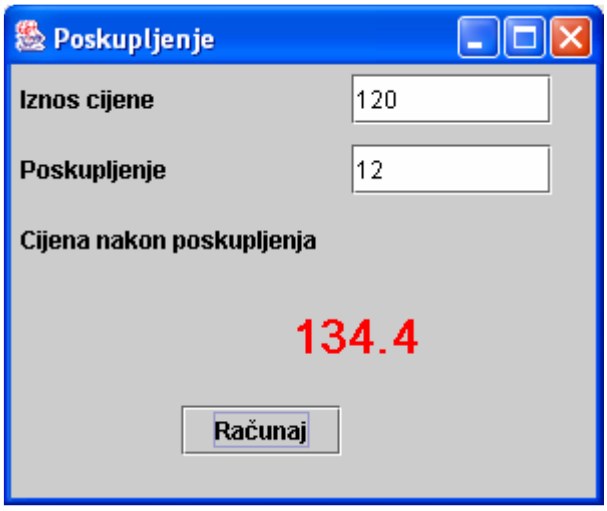
```
        }
        catch (Exception e1)
        {
                t2.setText (e1.getMessage ());
        }
}
```

Pokrenemo li sada program i u prvi okvir za tekst upišemo samo Ivan i razmak, program se ovaj puta neće srušiti, već će u drugom okviru za tekst pisati poruka: *Podaci nisu potpuni*

Zadaci za vježbu

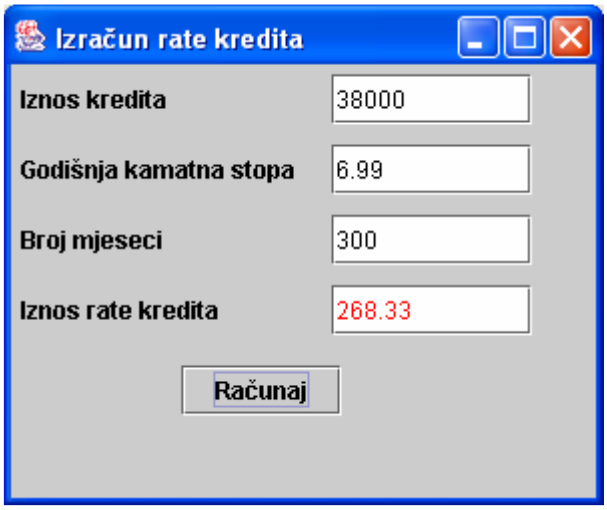
1. Objasni iznimke.
2. Navedi neke primjere koji mogu izazvati iznimku.
3. Kako procesiramo iznimke u Javi?
4. Navedi neke klase iznimaka u Javi.
5. Napiši program koji će crtati prozor kao na slici. Klikom na gumb **Novi broj** treba generirati i na labelu ispisati slučajan broj iz zadanog intervala.

6. Napiši program koji će crtati prozor kao na slici. Klikom na gumb **Računaj** na labelu se treba ispisati nova cijena koja se dobije uvećavanjem početne cijene za zadani postotak.



7. Napiši program koji će crtati prozor kao na slici. Klikom na gumb **Računaj** u okvir za tekst se treba ispisati iznos anuiteta kredita za upisani iznos kredita, kamatnu stopu te broj mjeseci. Iznos anuiteta se računa prema sljedećoj formuli:

$a = \frac{Cr^n(r-1)}{r^n - 1}$, pri čemu je: $r = 1 + \frac{p}{1200}$, $\boldsymbol{p}$ – godišnja kamatna stopa, $\boldsymbol{C}$ – iznos kredita i $\boldsymbol{n}$ – broj mjeseci otplate.



8. Napiši program koji će crtati prozor kao na slici. Klikom na gumb ***Mijenjaj*** boja teksta ***Tekst*** se treba promijeniti. Novu boja teksta će biti boja čije se RGB komponente unose u postavljene okvire za tekst.

9. Napiši program koji će crtati prozor kao na slici. U lijevom gornjem dijelu prozora ispisuje se zadatak. Zadatak se sastoji od dva broja i jedne od operacija (+, - ili *). Unosom broja u okvir za tekst i klikom na gumb ***Dalje*** treba provjeriti ispravnost rješenja i ovisno o točnom odgovoru izmijeniti broj točnih i broj ukupnih bodova (točno/ukupno) i generirati novi zadatak. Klikom na gumb ***Kraj*** treba završiti rad s programom.