Teorijske vežbe 9

Objektno-orjentisano programiranje

Izuzetak

- **Exception (exceptional event)**
  - Događaj koji narušava normalan tok izvršavanja programa i signalizira da se desila neka greška
- Dva tipa izuzetaka u Javi
  - **checked – izuzeci koji se moraju obavezno obraditi ili proslediti**
    - Program ne prolazi proces kompajliranja ako checked izuzetak nije obrađen/prosleđen
    - IOException, FileNotFoundException
  - **unchecked – izuzeci koji se ne moraju odbraditi**
    - Program prolazi proces kompajliranja ako unchecked izuzetak nije obrađen
    - ArrayIndexOutOfBoundsException, NegativeArraySizeException, NullPointerException, IllegalArgumentException, ArithmeticException, ClassCastException, NumberFormatException
- Svi izuzeci su klase koje nasleđuju (direktno ili indirektno) klasu Exception
  - Unchecked izuzeci su klase koje nasleđuju klasu RunTimeException koja nasleđuje klasu Exception

Obrada izuzetaka

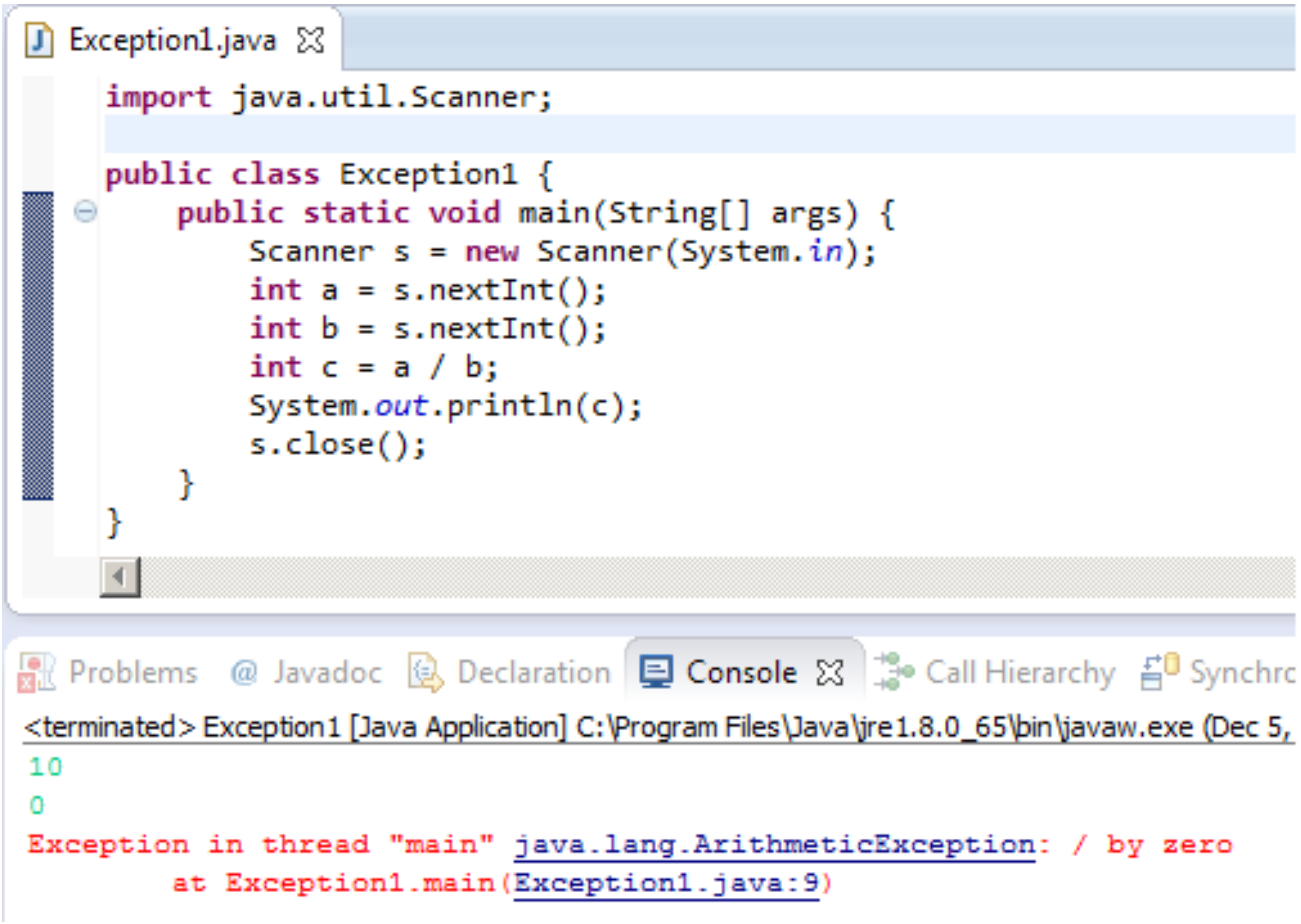
- try – catch – finally blok

```
try {
        // neki kod koji može da uzrokuje pojavu izuzetaka
        // ExceptionClass1, ExceptionClass2, … ExceptionClassN
} catch (ExceptionClass1 name1) {
        // obrada izuzetka ExceptionClass1
} catch (ExceptionClass2 name2) {
        // obrada izuzetka ExceptionClass2
}
…
} catch (EceptionClassN nameN) {
        // obrada izuzetka ExceptionClassN
} finally {
        // kod koji se izvršava na kraju desio se neki izuzetak ili ne
}
```

- Mora postojati bar jedan catch blok ili tačno jedan finally block. Catch blokova može biti i više
  - Redosled je bitan ako postoji hijerarhija nasleđivanja među izuzetcima
  - Od specifičnijih ka apstraktnijim izuzetcima
- Finally blok je opcion (ne mora postojati) ukoliko postoji bar jedan catch blok.

Unchecked izuzeci

- Unchecked izuzeci su uglavnom posledica semantičkih grešaka (bugova) u programu
- **Popraviti bug ako postoji, a ne obrađivati unchecked izuzetak**



Loše rešenje

Exception2.java

```
import java.util.Scanner;

public class Exception2 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        try {
            int a = s.nextInt();
            int b = s.nextInt();
            int c = a / b;
            System.out.println(c);
        } catch (ArithmeticException ae) {
            System.out.println("Deljenje nulom");
        }

        s.close();
    }
}
```

Problems @ Javadoc Declaration Console Call H

```
<terminated> Exception2 [Java Application] C:\Program Files\Java\jre1.8.0_65
10
0
Deljenje nulom
```

Dobro rešenje

```
import java.util.Scanner;

public class DobroResenje {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int a = s.nextInt();
        int b = s.nextInt();
        while (b == 0) {
            b = s.nextInt();
        }
        int c = a / b;
        System.out.println(c);
        s.close();
    }
}
```

Finally blok

Exception3.java

```
public class Exception3 {
    private static boolean simple() {
        try {
            int a = 10 / 0;
            System.out.println(a);
        } catch (ArithmeticException e) {
            System.out.println("Obrada greske... ");
            return false;
        } finally {
            System.out.println("Izvrsava se finally");
        }

        return true;
    }

    public static void main(String[] args) {
        System.out.println(simple());
    }
}
```

Problems @ Javadoc Declaration Console Call Hierarchy

```
<terminated> Exception3 [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javav
Obrada greske...
Izvrsava se finally
false
```

Upotreba finally bloka

- Osloboditi resurse i/ili reinicijalizovati promenljive desila se greška ili ne.

```
public static void printFile(String imeFajla) {
	BufferedReader br = null;
	try {
		br = new BufferedReader(new FileReader(imeFajla));
		String s;
		while ((s = br.readLine()) != null) {
			S.o.p(s);
		}
	} catch (FileNotFoundException fnfe) {
		S.o.p("Fajl " + imeFajla + " ne postoji");
	} catch (IOException ioe) {
		S.o.p("Greska prilikom citanja fajla " + imeFajla);
	} finally {
		if (br != null) {
			try {
				br.close();
			} catch (IOException ioe) {
				S.o.p("Greska kod zatvaranja fajla " + imeFajla);
			}
		}
	}
}
```

Pravljenje izuzetaka

- Možemo praviti naše izuzetke nasleđujući klasu Exception

```
public class MyException extends Exception {
	public MyException(String message) {
		super(message);
	}
}
```

Metoda može da generiše izuzetke

- Time se metodi koja ju je pozvala signalizira pojava greške
- Ključna reč **throw**

```
public void metod() throws MyException {
        …
        if (nešto nije kako valja)
            throw new MyException(“Nesto nije u redu”);
}
```

Obraditi ili proslediti izuzetak?

- Metoda može ne obraditi checked izuzetak ali ga mora proslediti
  - To smo do sada imali na više primera

```
public RadnaOrganizacija(String imeFajla) throws IOException {
    …
}
```

- Metoda A prosleđuje izuzetak metodi B koja je pozvala A samo ako B ume efektivnije i bolje da obradi grešku nego A

- Moguće je takođe i obraditi i proslediti izuzetak (rethrowing catched exception)

```
public void nekaMetoda() throws IOException {
    …
    try {
        // neki kod koji može da izazove IOException
    } catch (IOException ioe) {
        System.out.println(ioe.getMessage());
        throw ioe;
    }
    …
}
```

Izuzeci i petlje

Pretpostavimo da imamo sledeći kod

```
while (uslov) {
    blok1
    blok koji može prouzrokovati izuzetak
    blok2
}
```

Try-catch blok možemo postaviti na dva različita načina.

Izuzetci i petlje

```
while (uslov) {

    blok1

     try {
       // potencijalni izuzetak
     } catch (Izuzetak e) {
       // obrada greške
     }

     blok 2
}
```

Pojava izuzetka ne prekida petlju

```
try {
     while (uslov) {
        blok 1
        // potencijalni izuzetak
        blok 2
     }

} catch (Izuzetak e) {
    // obrada greške
}
```

Pojava izuzetka prekida petlju

Zadatak 1

- Putnički prtljag je predstavljen apstraktnom klasom Prtljag koja kao atribut ima težinu prtljaga.

- Klasa definiše konstruktor koji inicijalizuje sve atribute klase, get metode za sve atribute i apstraktnu metodu
  boolean izgubljen()
  koja kao rezultat vraća *true* ukoliko je prtljag zagubljen tokom transporta

- Neapstraktne klase RucniPrtljag i PredatiPrtljag nasleđuju klasu Prtljag.

- Rucni prtljag se ne može zagubiti, dok je verovatnoća da se zagubi predati prtljag 0.1

Zadatak 1

- Klasom Putnik je predstavljen jedan putnik na nekom avionskom letu. Svaki putnik ima ime, ručni prtljag i predati prtljag.

- Klasa definiše konstruktor koji inicijalizuje sve atribute klase i get metode za sve atribute klase.

- Klasom Let je predstavljen jedan avionski let. Kao atribute ova klasa ima naziv leta (String), nosivost aviona i niz putnika na letu. Klasa definiše konstruktor koji inicijalizuje sve atribute klase. **Ukoliko je ukupna težina putničkog prtljaga veća od nosivosti aviona konstruktor generiše izuetak VišakTereta.**

- Klasa Let takođe definiše metod int izgubljenPrtljag() koji kao rezultat vraća ukupnu težinu zagubljenog predatog prtljaga.

- Izuetak VišakTereta u konstruktoru prima naziv leta i formira poruku o grešci koja se prosleđuje konstruktoru nadklase.

Zadatak 1

- Klasom SpisakPutnika je predstavljen neki spisak (niz) putnika.

- Informacije o putnicima se čitaju iz ulaznog fajla koji je formatiran na sledeći način
  - Prva linija fajla – broj putnika
  - Svaka sledeća linija nosi informacije o jednom putniku i to ime, težinu ručnog prtljaga i težinu predatog prtljaga

- **Ne pretpostavljamo da je ulazni fajl dobro formatiran i obrađujemo sve greške u radu sa ulaznim fajlom.**

Zadatak 1

- Klasa SimulacijaLeta definiše main metod u kome se kreira spisak putnika iz nekog fajla, te se potom kreira let proizvoljne nosivosti koji treba da preveze putnike sa spiska. Na kraju metoda se štampa ukupna težina izgubljenog prtljaga.

Primer ulaznog fajla

7
mika, 8, 20
zika, 7, 21
pera, 5, 18
ana, 9, 23
zivana, 10, 15
mina, 0, 10
stojan, 5, 25

```java
public abstract class Prtljag {
	private int kilaza;

	public Prtljag(int kilaza) {
		if (kilaza < 0)
			throw new IllegalArgumentException("Negativna kilaza");

		this.kilaza = kilaza;
	}

	public int getKilaza() {
		return kilaza;
	}

	public abstract boolean izgubljen();
}
```

```java
public class RucniPrtljag extends Prtljag {
	public RucniPrtljag(int kilaza) {
		super(kilaza);
	}

	public boolean izgubljen() {
		return false;
	}
}


public class PredatiPrtljag extends Prtljag {
	public PredatiPrtljag(int kilaza) {
		super(kilaza);
	}

	public boolean izgubljen() {
		return Math.random() <= 0.1;
	}
}
```

```
public class VisakTereta extends Exception {
	public VisakTereta(String nazivLeta) {
		super("Visak tereta na letu " + nazivLeta);
	}
}

public class Let {
	private String nazivLeta;
	private int nosivost;
	private Putnik[] putnici;

	public Let(String nazivLeta, int nosivost, Putnik[] putnici)
		throws VisakTereta
	{
		this.nazivLeta = nazivLeta;
		this.nosivost = nosivost;
		this.putnici = putnici;

		int ukupnaTezina = 0;
		for (int i = 0; i < putnici.length; i++) {
			Putnik p = putnici[i];
			ukupnaTezina += p.getPredatiPrtljag().getKilaza();
			ukupnaTezina += p.getRucniPrtljag().getKilaza();
			if (ukupnaTezina > nosivost) {
				throw new VisakTereta(nazivLeta);
			}
		}
	}

	...
}
```

```
public int izgubljenPrtljag() {
	int ukupnaTezina = 0;

	for (int i = 0; i < putnici.length; i++) {
		Putnik p = putnici[i];
		PredatiPrtljag pp = p.getPredatiPrtljag();
		if (pp.izgubljen()) {
			ukupnaTezina += pp.getKilaza();
		}
	}

	return ukupnaTezina;
}

public String toString() {
	return nazivLeta + ", " + nosivost;
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class SpisakPutnika {
    private Putnik[] putnici;

    public boolean ucitajSpisak(String putniciFajl) {...

    public Putnik[] getPutnici() {
        return putnici;
    }
}
```

```
public boolean ucitajSpisak(String putniciFajl) {
	BufferedReader br = null;
	try {
		br = new BufferedReader(new FileReader(putniciFajl));
		int brojPutnika = Integer.parseInt(br.readLine());
		putnici = new Putnik[brojPutnika];
		for (int i = 0; i < brojPutnika; i++) {
			String linija = br.readLine();
			String[] tok = linija.split(",");
			if (tok.length == 3) {
				String ime = tok[0].trim();
				int tezinaRucni = Integer.parseInt(tok[1].trim());
				int tezinaPredati = Integer.parseInt(tok[2].trim());
				putnici[i] = new Putnik(ime,
					new RucniPrtljag(tezinaRucni),
					new PredatiPrtljag(tezinaPredati));
			} else {
				S.o.p("Fajl " + putniciFajl +
					" nije ispravno formatiran");
				return false;
			}
		}
		return true;
	} catch (NumberFormatException nfe) {
		S.o.p(nfe.getMessage()); return false;
	} catch (IllegalArgumentException iae) {
		S.o.p(iae.getMessage()); return false;
	} catch (IOException ioe) {
		S.o.p(ioe.getMessage()); return false;
	} finally {
		if (br != null) {
			try { br.close();
			} catch (IOException e) { System.out.println("..."); }
		}
	}
}
```

```
public class SimulacijaLeta {
	public static void main(String[] args) {
		SpisakPutnika sp = new SpisakPutnika();
		boolean ok = sp.ucitajSpisak("putnici.txt");
		if (ok) {
			try {
				Putnik[] p = sp.getPutnici();
				Let l = new Let("JU113", 200, p);
				S.o.p("Tezina izgubljenog prtljaga: " +
					l.izgubljenPrtljag());
			} catch (VisakTereta e) {
				S.o.p(e.getMessage());
			}
		}
	}
}
```

Zadatak 2

- Interfejs RadnaOrganizacija opisuje operacije nad jednom radnom organizacijom:
  - **void zaposli(String id, String ime, int plata) throws RadnaOrganizacijaException**
  - **void otpusti(String id) throws RadnaOrganizacijaException;**

- Metode zaposli i otpusti mogu generisati ROE ukoliko je došlo do neke greške prilikom zapošljavanja, otpuštanja radnika.
- Izuetak ROE mora čuvati informaciju o tome u kojoj od operacija je došlo do greške

Zadatak 2

- Klasom Firma je predstavljena jedna radna organizacija koja ima neki maksimalni broj zaposlenih radnika koji su predstavljeni nizom.
- Radnik u firmi je predstavljen ugnježdenom klasom koja ima sledeće atribute: id, ime, plata
- Napisati program koji simulira zapošljavanja i otpuštanja u nekoj firmi.

Zadatak 2

- Informacije o zapošljavanjima i otpuštanjima su date u ulaznom tekstualnom fajlu koji je organizovan na sledeći način
  - Jedna linija jedna akcija
  - Akcija kojom se opisuje zapošljavanje novog radnika je u formatu
    “zaposli, <ID>, <ImeRadnika>, <PlataRadnika>
  - Akcija kojom se opisuje otpuštanje novog radnika je u formatu

    “otpusti, <ID>”

Ne pretpostavljamo da je ulazni fajl ispravno formatiran, ali sprovodimo sve one akcije koje su dobro formatirane

Primer ulaznog fajla

otpusti, 12
zaposli, 1, Mika, 10000
otpusti, 5
zaposli, 2, Pera, 12000
zaposli, 1, Zika, 10000
zaposli, 3, Stojan, 8000
otpusti, 1
zaposli, 4, Milovan, 9004f
zaposli, 4, Milovan, 9000
zaposli, 5, Ana, 13000
zaposli, 6, Mina, 15000
zaposli, 7, Dara, 23000
zaposli, 8, Sara, 12000
Ana, voli, Milovana
zaposli, 9, Mara, 12000, Sara
zaposli, 9, Mara, 12000
zaposli, 10, Ana, 5000
zaposli, 11, Zivka, 4000
otpusti, 3
zaposli, 11, Zivka, 4000

```
public interface RadnaOrganizacija {

    void zaposli(String id, String ime, int plata)
        throws RadnaOrganizacijaException;

    void otpusti(String id)
        throws RadnaOrganizacijaException;
}
```

```java
public class RadnaOrganizacijaException extends Exception {
	public enum TipGreske {
		ZAPOSLJAVANJE,
		OTPUSTANJE
	}

	private TipGreske t;
	private String opis;

	public RadnaOrganizacijaException(TipGreske t, String opis) {
		super(opis);
		this.opis = opis;
		this.t = t;
	}

	public TipGreske getTipGreske() {
		return t;
	}

	public String opisGreske() {
		if (t == TipGreske.ZAPOSLJAVANJE) {
			return "Greska kod zaposljavanja: " + opis;
		} else {
			return "Greska kod otpustanja: " + opis;
		}
	}
}
```

```
public class Firma implements RadnaOrganizacija {

    private class Radnik {..

    private int maxZaposlenih;
    private int brojZaposlenih;
    private Radnik[] radnici;

    public Firma(int maxZaposlenih) {..

    private int pretrazi(String id) {..

    public void zaposli(String id, String ime, int plata) ..

    public void otpusti(String id) ..

    public void stampajSve() {..
}
```

```
private class Radnik {
	String id, ime;
	int plata;

	public Radnik(String id, String ime, int plata) {
		this.id = id;
		this.ime = ime;
		this.plata = plata;
	}

	public String toString() {
		return id + ", " + ime + ", " + plata;
	}
}
```

```
public Firma(int maxZaposlenih) {
	this.maxZaposlenih = maxZaposlenih;
	brojZaposlenih = 0;
	radnici = new Radnik[maxZaposlenih];
}

private int pretrazi(String id) {
	for (int i = 0; i < brojZaposlenih; i++) {
		if (radnici[i].id.equals(id)) {
			return i;
		}
	}

	return -1;
}

public void stampajSve() {
	System.out.println("Zaposleni u radnoj organizaciji: ");
	for (int i = 0; i < brojZaposlenih; i++) {
		System.out.println(radnici[i]);
	}
}
```

```
public void zaposli(String id, String ime, int plata)
	throws RadnaOrganizacijaException
{
	if (brojZaposlenih == maxZaposlenih) {
		throw new RadnaOrganizacijaException(
			RadnaOrganizacijaException.TipGreske.ZAPOSLJAVANJE,
			"Firma vec ima maksimalan dozvoljen broj radnika"
		);
	}

	if (pretrazi(id) != -1) {
		throw new RadnaOrganizacijaException(
			RadnaOrganizacijaException.TipGreske.ZAPOSLJAVANJE,
			"Radnik sa identifikatorom " + id + " vec postoji"
		);
	}

	radnici[brojZaposlenih++] = new Radnik(id, ime, plata);
}
```

```
public void otpusti(String id)
	throws RadnaOrganizacijaException
{
	if (brojZaposlenih == 0) {
		throw new RadnaOrganizacijaException(
			RadnaOrganizacijaException.TipGreske.OTPUSTANJE,
			"Firma nema zaposlenih"
		);
	}

	int pozicija = pretrazi(id);
	if (pozicija == -1) {
		throw new RadnaOrganizacijaException(
			RadnaOrganizacijaException.TipGreske.OTPUSTANJE,
			"Radnik sa identifikatorom " + id + " ne postoji"
		);
	}

	for (int i = pozicija; i < brojZaposlenih - 1; i++) {
		radnici[i] = radnici[i + 1];
	}
	--brojZaposlenih;
}
```

```
public class SimulacijaFirme {
	public static void main(String[] args) {
		BufferedReader br = null;
		try {
			Firma firma = new Firma(10);
			br = new BufferedReader(
				new FileReader("FirmaAkcije.txt"));
			String linija;
			int brLinija = 0;

			while ((linija = br.readLine()) != null) {
				++brLinija;
				boolean ok = obradiAkciju(firma, linija);
				if (!ok) {
					S.o.p("Greska u liniji " + brLinija);
				}
			}

			firma.stampajSve();
		} catch (IOException e) {
			System.out.println("Greska u radu sa ulaznim fajlom: ");
			System.out.println(e.getMessage());
		} finally {
			if (br != null) {
				try {
					br.close();
				} catch (IOException e) {
					System.out.println(e.getMessage());
				}
			}
		}
	}
```

```
private static boolean obradiAkciju(Firma firma, String akcija) {
	String[] t = akcija.split(",");
	String tipAkcije = t[0].trim();
	if (tipAkcije.equals("zaposli")) {
		if (t.length != 4) {
			S.o.p("Neodgovarajuci opis akcije zaposli");
			return false;
		}
		try {
			String id = t[1].trim();
			String ime = t[2].trim();
			int plata = Integer.parseInt(t[3].trim());
			firma.zaposli(id, ime, plata);
		} catch (NumberFormatException nfe) {
			S.o.p("Plata nije integer");
			return false;
		} catch (RadnaOrganizacijaException roe) {
			S.o.p(roe.opisGreske());
			return false;
		}
		return true;
	} else if (tipAkcije.equals("otpusti")) {
		if (t.length != 2) {
			S.o.p("Neodgovarajuci opis akcije otpusti");
			return false;
		}
		String id = t[1].trim();
		try {
			firma.otpusti(id);
		} catch (RadnaOrganizacijaException roe) {
			S.o.p(roe.opisGreske());
			return false;
		}
		return true;
	} else {
		S.o.p("Nepostojeci tip akcije: " + akcija);
		return false;
	}
}
```