LINKgroup

Distance Learning System

Core Java Programming

Obrada i generisanje izuzetaka

Greške u aplikaciji

- Greške u aplikacijama, mogu se podeliti na dva tipa:
  - **Sintaksne** (greška u sintaksi, lako se ispravlja)
  - **Logičke** (logička greška, teže se ispravlja)
- Takođe, greške se mogu podeliti po trenutku događajanja:
  - **Greške u prevođenju**
  - **Greške u izvršavanju (runtime greške)**

Izuzeci

- Greške u izvršavanju, događaju se najčešće zbog nedostatka resursa, neispravnog indeksiranja, konverzije i slično
- Mi, kao developeri, ove greške moramo predvideti i sprečiti
- Java sadrži mehanizam koji omogućava da aplikacija na „miran" način objavi kako nešto nije u redu, dajući developeru mogućnost da u tom trenutku reaguje. Ovaj mehanizam je sistem **izuzetaka**.
- Izuzetak je situacija u kojoj neki deo programa nije u stanju da funkcioniše po predviđenom planu, ali, za razliku od greške, dozvoljava programu da izvrši alternativu koja će sanirati problem i sprečiti prekid rada programa

Primer izuzetka (jcex122014 JavaExceptions)

- Ako bi pokušali da startujemo Java program:

```
package javaexceptions;
public class JavaExceptions {
    public static void main(String[] args) {
        int x = 100/0;
    }
}
```

Izlaz bi bio sledeći:

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at javaexceptions.JavaExceptions.main(JavaExceptions.java:4)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Odnosno, Java bi nam dala do znanja da nešto nije u redu, emitovala grešku i prekinula rad aplikacije
- Ali ovu grešku nije izazvala pogrešna aritmetička operacija, već **neobrađen izuzetak**

Tipovi obrade izuzetka

- U prethodnom primeru Java je prepoznala nepravilnost u vrednostima promenljivih, i na osnovu te nepravilnosti generisala **Exception** objekat, a zatim ga isporučila aplikaciji
- Exception objekat „isplivava" na površinu aplikacije u trenutku kada se pojavi u sistemu, i ako se niko ne „pozabavi" njime (obradi ga), isplivaće na samu površinu i aplikacija će prestati sa radom
- Za obradu izuzetka koristimo jednu od dve tehnologije:
  - Slanje izuzetka dalje
  - „Hvatanje" izuzetka

Hvatanje izuzetka

- Najčešća situacija obrade izuzetka je njegovo „hvatanje“
- Za hvatanje izuzetaka koriste se **try / catch** blokovi
- Try catch blokovi funkcionišu po sledećem principu:
  - Kreira se struktura, sa minimum dva bloka
    try { } catch (Exception ex) { }
  - Deo koda u kome se očekuje izuzetak se smešta u **try** blok
  - Deo koji bi trebalo da se izvrši ukoliko ne uspe izvršavanje prvog bloka smešta se u **catch** blok
  - Cela struktura, sintaksno izgleda ovako:

```
try {
    int x = 100/0;
} catch(Exception ex){
    System.out.println("Hey, you can't divide by zero!");
}
```

Bočni efekti u try / catch bloku

- Primer sa prethodnog slajda sprečava pojavljivanje greške, i ne prekida izvršavanje programa. Ipak, problem i dalje postoji, i u trenutku kada dođe do pokušaja deljenja sa nulom, kompletna grana programa će biti prekinuta (ako postoji finally block, on se izvršava i u ovom slučaju)

U ovom trenutku grana se prekida

A zatim se ovde nastavlja

Nikada se ne izvrši

```
int y = 5;
try {
    int x = 100/0;
    y = 10;
} catch(Exception ex){
    System.out.println("Hey, you can't divide by zero!");
}
System.out.println(y);
```

- Kolika će biti vrednost promenljive y nakon try / catch bloka?

Vežba

- Potrebno je kreirati program koji sabira brojeve, tako što konstantno traži od korisnika da unese prvi pa drugi operand
- Ukoliko korisnički unos nije u redu, program prikazuje poruku da promenljive nisu ispravne, a zatim nudi korisniku ponovni unos
- Program NE mora da nakon greške nastavi sa započetom operacijom, već treba da počne sledeću (unos oba operanda iznova)

Tipovi izuzetaka

- Nakon try bloka, sledi catch blok. Ovaj blok je parametrizovan za razliku od try bloka i kao parametar prihvata objekat klase očekivanog izuzetka:

```
} catch(Exception ex){
```

- Ovo ne može biti bilo koja klasa već isključivo klasa Exception ili klasa koja je nasleđuje

```
} catch(Object ex){
```

- Takođe je važno da parametar catch bloka odgovara očekivanom izuzetku, inače on neće ni biti uhvaćen. Na primer, sledeći kod će izbaciti grešku:

```
try {
    int x = 100/0;
} catch(ArrayStoreException ex){
    System.out.println("Hey, you can't divide by zero!");
}
```

Multiple catch blokovi

- Videli smo da ArrayStoreException objekat nije ispravno reagovao na aritmetički izuzetak, ali da Exception objekat jeste. Ovo se događa zbog toga što se izuzeci hvataju po nivou specijalizacije.
- Klasa Exception je u samom vrhu svih klasa izuzetaka, i zbog toga „hvata" sve izuzetke. Prilikom deljenja sa nulom, dolazi do specijalizovanog izuzetka: **ArithmeticException**, koji je čak dva koraka u hijerarhiji od **Exception** klase.
- **ArrayStoreException** je takođe specijalizovana izuzetak klasa, ali ne odgovara klasi ArithmeticException, i zato izuzetak nije adekvatno uhvaćen
- Java omogućava provere izuzetaka različitog tipa, pomoću multiple catch blokova

```
try {
    int x = 100/0;
} catch(ArrayStoreException ex){
    System.out.println("I will never be catched");
}  catch(ArithmeticException ex){
    System.out.println("Hey, you can't divide by zero!");
}
```

Multiple catch blokovi

- Multiple catch blokovi se obično formiraju tako da se izuzeci „hvataju" od specijalnih ka generalnim. Dobra praksa je da se na kraju svih specijalnih slučajeva, konačno uhvati i Exception, koji će uhvatiti sve izuzetke koji su eventualno prošli filtraciju

```
try {
    Object[] array = new Integer[3];
    array[0] = "Hello";
} catch(ArrayStoreException ex){
    System.out.println("I will never be catched");
}  catch(ArithmeticException ex){
    System.out.println("Hey, you can't divide by zero!");
}  catch(Exception ex){
    System.out.println("I am here just in case");
}
```

Finally blok

- Osim try i catch, postoji još jedan, opcioni blok unutar try catch strukture. To je blok **Finally**
- Finally blok se izvršava u bilo kom slučaju. Gotovo isto kao da smo napisali kod izvan kompletnog try catch bloka
- Ovaj blok koristi se najčešće za raspuštanje resursa
- Finaly blok nema parametre

```
try {
    Object[] array = new Integer[3];
    array[0] = "Hello";
} catch(ArrayStoreException ex){
    System.out.println("I will never be catched");
}  catch(ArithmeticException ex){
    System.out.println("Hey, you can't divide by zero!");
}  catch(Exception ex){
    System.out.println("I am here just in case");
} finally {
    System.out.println("I will be executed anyway");
}
```

Ručno izbacivanje izuzetaka
(jcex122014 ManualExceptionThrowing)

- Izuzetak možemo izbaciti i ručno, u bilo kom trenutku izvršavanja programa
- Izuzetak se ručno može izbaciti naredbom **throw**.
- Naredbi throw mora uslediti validan Exception objekat

```
Exception ex = new Exception();
throw ex;
```

ili samo

```
throw new Exception();
```

Ručno izbacivanje izuzetaka

- U primeru, kreiran je metod za deljenje brojeva do deset
- Ovaj metod nema problem sa nulom kao deliocem, tada jednostavno vraća kao rezultat broj 0, ali ima problem sa operandima koji su veći od 10
- Pošto Java nema problem sa deljenjem brojeva većih od deset, neće izbaciti nikakav izuzetak ukoliko ubacimo takav broj u metod, pa zato moramo ovu situaciju obraditi ručno

```
package manualexceptionthrowing;
public class ManualExceptionThrowing {
    static int divide(int a, int b){
        if(b==0){
            return 0;
        } else
        if(a>10||b>10){
            throw new ArithmeticException("Larger than 10");
        } else {
            return a+b;
        }
    }
    public static void main(String[] args)  {
        System.out.println(divide(14,1));
    }
}
```

```
run:
Exception in thread "main" java.lang.ArithmeticException: Larger than 10
        at manualexceptionthrowing.ManualExceptionThrowing.divide(ManualExceptionThrowing.java:8)
        at manualexceptionthrowing.ManualExceptionThrowing.main(ManualExceptionThrowing.java:14)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Struktura klase Exception

- Pojavljivanje izuzetka ukazuje na to da je došlo do nepravilnosti u programu. Ali takođe, objekat klase izuzetak nosi u sebi dodatne informacije vezane za izazvanu nepravilnost
- Ove informacije možemo dobaviti putem geter metoda izuzetka ili metode **printStackTrace**:

```
try {
 System.out.println(divide(14,1));
} catch(ArithmeticException ex){
    System.out.println(ex.getCause());
    System.out.println(ex.getClass());
    System.out.println(ex.getMessage());
    ex.printStackTrace();
}
```

Korisnički definisani izuzeci
(jcex122014 CustomExceptions)

- Možemo kreirati sopstveni izuzetak, dovoljno je da mu damo ime, i nasledimo klasu Exception

```
public class MyException extends Exception { }
```

- Ukoliko postoji potreba, ponašanje klase može se modifikovati prepisivanjem metoda

```
public class MyException extends Exception {
    @Override
    public String toString(){
        return "Something nice";
    }
    @Override
    public String getMessage() {
        return "Some nice custom message";
    }
}
```

Proveravani i neproveravani izuzeci

- Izuzeci koji nasleđuju klasu Exception su **proveravani** izuzeci. Eventualnu pojavu ovih izuzetaka moramo obraditi
- **Neproveravani** izuzeci nasleđuju klasu **Error**, i njih ne moramo obrađivati ako ne želimo

```
package customexceptions;
public class MyUncheckedException extends Error { }

public class CustomExceptions {
    static void throwingUnchecked(){
        throw new MyUncheckedException();
    }
    public static void main(String[] args) {
        throwingUnchecked();
```

Proveravani i neproveravani izuzeci

```
public class CustomExceptions {
    static void throwingUnchecked(){
        throw new MyUncheckedException();
    }
    static void throwingChecked() throws MyException{
        throw new MyException();
    }
    public static void main(String[] args) {
        throwingUnchecked();
        try {
            throwingChecked();
        } catch (MyException ex){ }

        try {
            throw new MyException();
        } catch(MyException ex){
            System.out.println("Hey, this is same as regular Exception");
            System.out.println("Custom exception text: " + ex);
        }
    }
```

- Nije obavezno hvatanje
- Obavezno hvatanje

Slanje izuzetka dalje

- Izuzeci se dešavaju isključivo unutar metode
- Ako ne želimo da obrađujemo izuzetak u metodi - ne moramo, ali to će imati posledice na deo programa koji je aktivirao metodu
- Nakon unosa ovakve linije u kod, moramo obavestiti ostatak sistema (deo koji poziva metodu u kojoj se kod nalazi), da postoji potencijalno izbacivanje izuzetka Oznakom **throws Exception** u potpisu metode (Exception može biti bilo koji tip Exceptiona, ili više njih, odvojenih zarezima)

```
static void throwingChecked() throws MyException{
    throw new MyException();
}
```

Vežba 1 (jcex122014 CustomExceptions)

- Postojeću aplikaciju potrebno je obezbediti tako da ne prijavljuje grešku

```
public class Calculator {
    static int calculate(int a, int b, String op)
    {
        if(op.equals("+"))
                return a+b;
        if(op.equals("-"))
                return a-b;
        if(op.equals("/"))
                return a/b;
        if(op.equals("*"))
                return a*b;
        return 0;
    }
    public static void main(String[] args){
            int x = calculate(5,0,"/");
            System.out.println(x);
    }
}
```

Vežba 2 (jcex122014 User)

Postoji sledeća klasa User:

```
package user;
public class User {
    public int id;
    public String firstName;
    public String lastName;
    public String email;
    public User(int id, String firstName, String lastName, String email)
    {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }
}
```

- Potrebno je kreirati klasne izuzetke za nepravilan unos ID-a, imena, prezimena i E-mail-a.
- Potrebno je implementirati sistem provere u konstruktor klase User tako da ukoliko je ID veći od 100, bude izbačen InvalidIdException, ako su firstName, lastName i E-mail polja prazna, bude izbačen InvalidFirstNameException, InvalidLastNameException ili InvalidEmailException.
- Potrebno je instancirati ovu klasu u Main projektu.