IZUZECI

2016/17

PRIRODNO-MATEMATIČKI FAKULTET, UNIVERZITET U KRAGUJEVCU

GREŠKE

- Tokom izvršavanja aplikacije javljaju se greške različitih nivoa ozbiljnosti.
- Pri pozivu nekog metoda, moguće je da dođe do različith vrsta neregularnih stanja. Na primer:
  - prosleđeni argumenti nisu odgovarajući
  - problem internog stanja objekta kome je poruka poslata (nekonzistentne vrednosti polja)
  - greške na resursima ili podacima kojima metod manipuliše (npr. u fajlu ili mrežnoj adresi)
- Takve situacije mogu manje ili više uticati na tok izvršavanja programa, u svakom slučaju tok programa se razlikuje od regularnog, tj. predstavlja izuzetak od pravilnog.

OBRADA GREŠAKA

- Jedan način rukovanja takvim (izuzetnim) situacijama jeste proveravanje svih kritičnih uslova pre navodjenja dela koda koji može biti 'ugrožen' neproverenim greškama.

```
GET A FILENAME
OPEN THE FILE
IF THERE IS NO ERROR OPENING THE FILE
    READ SOME DATA
    IF THERE IS NO ERROR READING THE DATA
        PROCESS THE DATA
        WRITE THE DATA
        IF THERE IS NO ERROR WRITING THE DATA
            CLOSE THE FILE
            IF THERE IS NO ERROR CLOSING FILE
                RETURN
```

- Mana ovakvog rada jeste nečitak kod u kome se mešaju
  - provera uslova za regularan tok programa i obrada u slučaju neispunjenosti uslova
  - kod koji se izvršava pri regularnom radu

RAZDVAJANJE REGULARNOG KODA OD OBRADE IZUZETNIH SITUACIJA

Da bi obezbedili razdvajanje koda koji se izvršava kada program teče glatko od obrade 'neregularne' situacije objektni jezici uvode:

1. posebnu vrstu upravljačkog bloka – `try/catch/finally` i
2. poseban mehanizam obaveštavanja da se i šta desilo – instanciranjem objekata specijalnog tipa i njegovim prosleđivanjem onom delu koda koji je u stanju da ga "obradi".

```
TRY TO DO THESE THINGS:
     GET A FILENAME
     OPEN THE FILE
     READ SOME DATA
     PROCESS THE DATA
     WRITE THE DATA
     CLOSE THE FILE
     RETURN
IF THERE WAS AN ERROR OPENING THE FILE THEN DO ...
IF THERE WAS AN ERROR READING THE DATA THEN DO ...
IF THERE WAS AN ERROR WRITING THE DATA THEN DO ...
IF THERE WAS AN ERROR CLOSING THE FILE THEN DO ...
```

```
try {
                . . .
}
catch (ExceptionType1 exceptionObject) {
     // obrada izuzetka tipa 1
}
catch (ExceptionType2 exceptionObject) {
     // obrada izuzetka tipa 2
}
finally  {
     . . .
}
```

PRIMER 1 - BEZ OBRADE GREŠAKA

```
class  Ssluzba { . . .
void ispraviOcenu(Prijava p,int nova){
     p.ocenaIspravka(nova);

}
}
```

```
main() {
     Ssluzba s; Prijava p;
     . . .
     s.ispraviOcenu(p,5);
     . . .
}
```

Jedno rešenje – uvesti povratnu vrednost za metod, pa je onda ispitivati pri pozivu f-je.
U tom slučaju se regularan od neregularnog koda razdvaja i if/else granama.

```
class Prijava{
int ocena; Student s; Predmet p;
. . .
     void ocenaIspravka (int nova) {
          if !(p.ocena()>5 && nova>5 && nova<=10) {
               // nije moguce iyvrsiti ispravku
          }
          else p.setOcena(nova);
     }
}
```

POJEDNOSTAVLJEN MODEL OBRADE GREŠAKA UPOTREBOM KONCEPTA IZUZETAKA – PRIMER I

```
main() {
     Ssluzba s; Prijava p;
     . . .
     s.ispraviOcenu(p,5);
     . . .
}
```

POJEDNOSTAVLJEN MODEL OBRADE GREŠAKA UPOTREBOM KONCEPTA IZUZETAKA – PRIMER I

```
class  Ssluzba { . . .
void ispraviOcenu(Prijava p,int nova){
try  {    // NADGLEDANI REGION
     . . .
     p.ocenaIspravka(nova);
     . . .
  }
  catch(Greska g) {
      // obrada greske
  }
//DEO KODA KOJI SE IZVRSAVA BEZ OBZIRA NA TRY/CATCH
}
}
```

```
main() {
     Ssluzba s; Prijava p;
     . . .
     s.ispraviOcenu(p,5);
     . . .
}
```

POJEDNOSTAVLJEN MODEL OBRADE GREŠAKA UPOTREBOM KONCEPTA IZUZETAKA – PRIMER I

```
class  Ssluzba { . . .
void ispraviOcenu(Prijava p,int nova){
try  {    // NADGLEDANI REGION
     . . .
     p.ocenaIspravka(nova);
     . . .
  }
  catch(Greska g) {
      // obrada greske
  }
//DEO KODA KOJI SE IZVRSAVA BEZ OBZIRA NA TRY/CATCH
}
}
```

```
main() {
     Ssluzba s; Prijava p;
     . . .
     s.ispraviOcenu(p,5);
     . . .
}
```

```
class Prijava{
int ocena; Student s; Predmet p;
. . .
     void ocenaIspravka throws Greska (int nova) {
          if !(p.ocena()>5 && nova>5 && nova<=10) {
          Greska g = new Greska();
          throw g;
          }
          else p.setOcena(nova);
     }
}
```

POJEDNOSTAVLJEN MODEL OBRADE GREŠAKA UPOTREBOM KONCEPTA IZUZETAKA – PRIMER I

```
class  Ssluzba { . . .
void ispraviOcenu(Prijava p,int nova){
try  {    // NADGLEDANI REGION
     . . .
     p.ocenaIspravka(nova);
     . . .
  }
  catch(Greska g) {
       // obrada greske
  }
//DEO KODA KOJI SE IZVRSAVA BEZ OBZIRA NA TRY/CATCH
}
}
```

```
main() {
     Ssluzba s; Prijava p;
     . . .
     s.ispraviOcenu(p,5);
     . . .
}
```

Delovi koda koji neće biti izvršeni

```
class Prijava{
int ocena; Student s; Predmet p;
. . .
     void ocenaIspravka throws Greska (int nova) {
          if !(p.ocena()>5 && nova>5 && nova<=10) {
          Greska g = new Greska();
          throw g;
          }
          else p.setOcena(nova);
     }
}
```

POJEDNOSTAVLJEN MODEL OBRADE GREŠAKA UPOTREBOM KONCEPTA IZUZETAKA – PRIMER I

Ko obaveštava?

Metod tokom čijeg izvršavanja se desila izuzetna situacija, koja zahteva drugačije ponašanje pozivaoca metoda koji izuzetak baca.

Kako obaveštava?

Generiše objekat specijalnog tipa (tipa IZUZETKA) i baca ga, tj. prosleđuje ga komandi `throw`.

Ko preuzima obaveštenje?

`throw` JVM izvršava tako što preusmerava izvršavanje programa na deo koda koji je naveden u bloku za obradu odgovarajućeg tipa izuzetka, tj. za prihvatanje objekata odgovarajućeg tipa (`catch` blok).

Koji je to specijalni tip objekta?

Objekat kojim se opisuje greška ne može pripadati proizvoljnoj klasi. Klasa kojoj objekat pripada mora da se nađe u lancu nasleđivanja klase `Throwable`, jer sam Javin kompajler vrši proveru tipa objekta pri prevođenju `throw` komande.

Šta je sa nenadgledanim delom koda (van `try` bloka)?

Ako dodje do bilo kakve greške van nadgledanog regiona, sigurno je da metod u kom se greška desila neće istu zbrinuti, već će greška biti delegirana njegovom pozivaocu.

POJEDNOSTAVLJEN MODEL OBRADE GREŠAKA UPOTREBOM KONCEPTA IZUZETAKA – PRIMER 1

Zašto se neobrađen izuzetak po automatizmu prosleđuje pozivaocu?

Zato što izuzetak, ako je već generisan, mora da bude obrađen. Njegovo bacanje izaziva neregularan završetak metoda u kome se javio, pa aplikacija mora da ima predviđenu reakciju na takvu situaciju.
Ako izuzetak ne obradi prvi metod kome je izuzetak prosleđen, onda se izuzetak delegira dalje, tj. njegovom pozivaocu (sve do `main()` metoda).

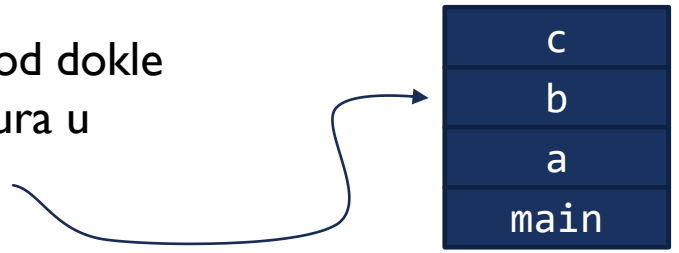
PRIMER 1. komentar

Klasa Prijava ima predviđen metod prepravke ocene, ali samo u slučaju da u prijavi piše da je student položio i ako je nova ocena između 6 i 10, tj. Ako se ne menja status 'položenosti' ispita. U suprotnom se metod u kom se poziva prepravka ocene obaveštava o tome da ispravka ocene nije obavljena tako što se baca izuzetak.
Dakle, neuspešna ispravka je logički svrstana u izuzetak i to u funkcionisanju pozivaoca. Sama prijava ne zavisi od toga, tj. Ona se brine o konzistentnosti svoga stanja, a onoga koji je pokušao da je u nekonzistentno stanje dovede, obaveštava o neuspehu njegove akcije. Da li će i kako pozivač reagovati nije briga prijave, već onoga ko izuzetak treba da uhvati.

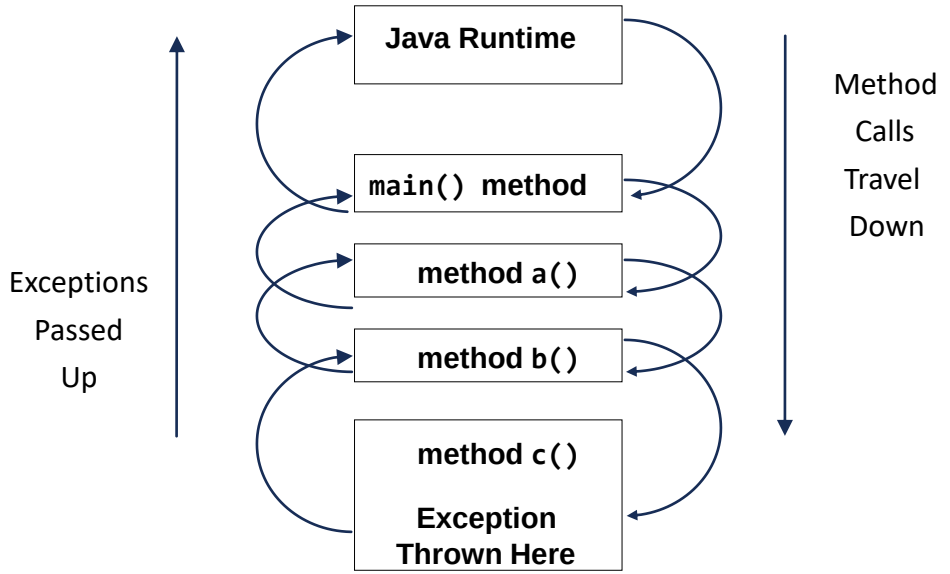
LANAC HVATANJA IZUZETKA I *CALL STACK*

Pretpostavimo da se u `main()` metodu u nekom trenutku poziva metod `a()` u kome se poziva metod `b()`, a u njemu metod `c()`.

O čitavom lancu pozivanja i o tome koji je metod dokle stigao sa izvršavanjem vodi računa JVM. Struktura u kojoj beleži lanac pozivanja se naziva call stack.

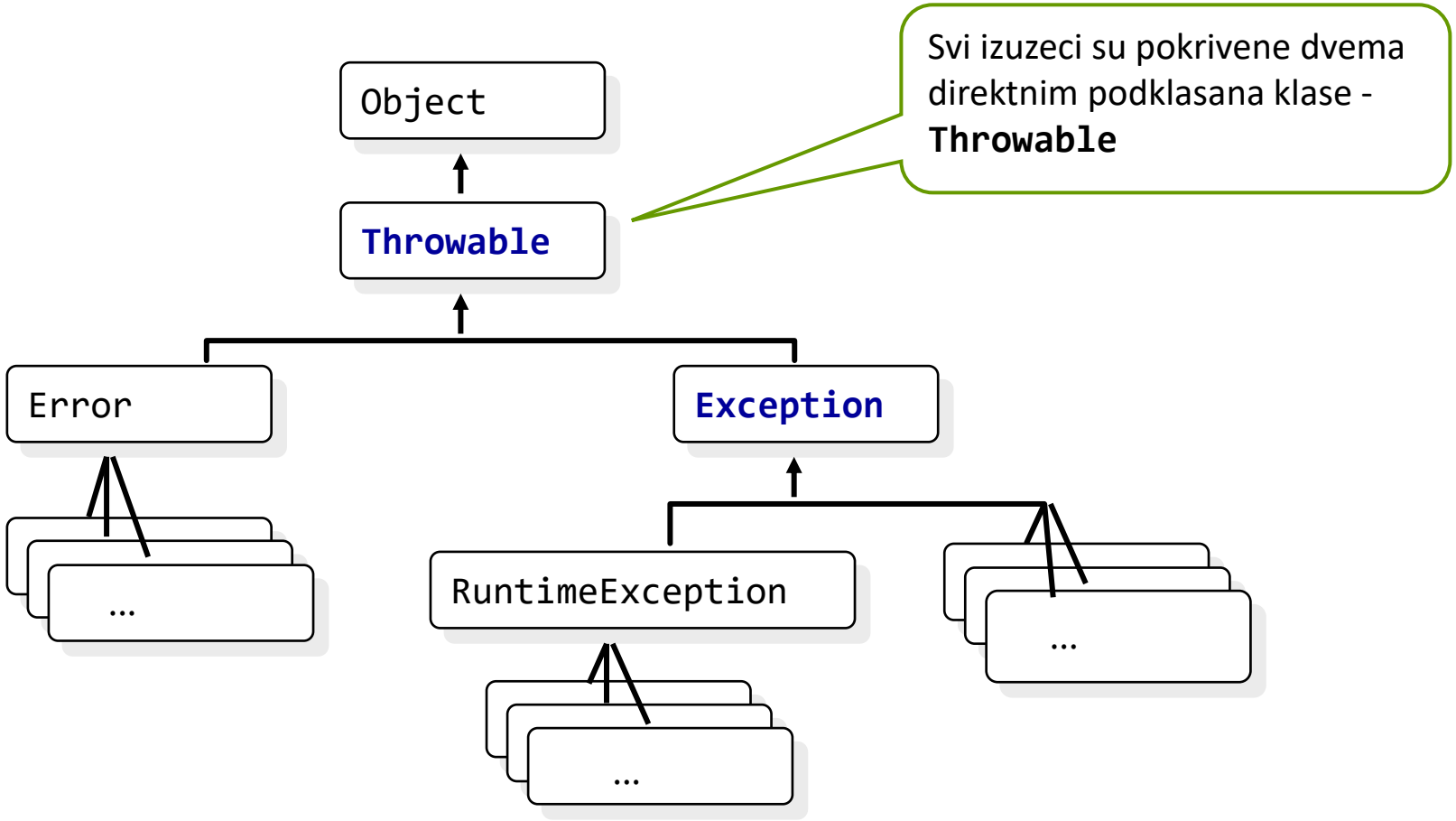


Zahvaljujući call stack-u JVM zna koja je prva naredna komanda koja treba da bude izvršena nakon regularnog završetka nekog pozvanog metoda ili kome treba da prosledi generisani izuzetak u slučaju neregularnog završetka pozvanog metoda.



Ako JVM ne uspe ni u jednom metodu na call stack-u da pronađe obradu izuzetka program prekida rad.

TIPOVI IZUZETAKA U JAVI



IZUZECI TIPA ERROR

- Predstavljaju izuzetke za koje se ne očekuje da ih hvata korisnik.
- Tri direktne podklase klase Error:
  - `ThreadDeath`: baca se kada se namerno zaustavi nit u izvršavanju. Kada se ne uhvati završava nit, a ne program.
  - `LinkageError`: ozbiljne greške unutar klasa u programu (nekompatibilnosti među klasama, pokušaj kreiranja objekta nepostojeće klase)
  - `VirtualMachineError` – JVM greška

IZUZECI TIPA `Exception` I, POSEBNO, `RuntimeException`

- Za skoro sve izuzetke koji su obuhvaćeni podklasama klase `Exception` potrebno je uključiti kod za njihovu obradu ili program neće proći kompajliranje, tj. prevodilac NE DOZVOLJAVA njihovo ignorisanje.
- `RunTimeException` izuzeci se tretiraju drugačije, jer se u njih najčešće svrstavaju izuzeci koji se pojavljuju kao posledica ozbiljnijih grešaka u kodu, čija obrada ne bi mogla ništa značajno promenila. Prevodilac dozvoljava njihovo ignorisanje.
- Neke podklase klase `RunTimeException`:
  - `ArithmeticException`: neispravan rezultat aritmetičke operacije poput dijeljenja nulom.
  - `IndexOutOfBoundsException`: indeks koji je izvan dozvoljenih granica za objekat poput niza, stringa ili vektora.
  - `NegativeArraySizeException`: upotreba negativnog indeksa niza.
  - `NullPointerException`: poziv metoda ili pristup podatku članu null objekta.
  - `ArrayStoreException`: pokušaj dodeljivana reference pogrešnog tipa elementu niza.
  - `ClassCastException`: pokušaj kastovanja objekta neodgovarajućeg tipa.

PROVERAVANI I NEPROVERAVANI IZUZECI

Na osnovu toga da li prevodilac insistira njihovom proveravanju ili ne, izuzeci se dele u dve grupe:

- **proveravani izuzeci (checked)**
  Oni koji su izvedeni iz klase Exception i svi koji nisu u lancu RunTimeException klase.
  Ako metoda baca neki proveravani izuzetak, poziv te metode mora da bude uokviren try-catch blokom koji hvata taj izuzetak, a metoda mora da bude označena ključnom reči throws i nazivom klase izuzetka koji baca.
- **neproveravani izuzeci (unchecked)**
  Oni koji su izvedeni iz RunTimeException.Klase koje su navedene u tabeli Javinih predefinisanih izuzetaka, uglavnom, nasleđuju klasu RuntimeException pa pripadaju grupi neproveravanih izuzetaka.
  Ako metoda baca neki neproveravani izuzetak, poziv te metode može, ali ne mora biti uokviren try/catch blokom koji hvata taj izuzetak.

NEPROVERAVANI IZUZETAK - PRIMER

```
public class Izuz extends RunTimeException{
    ...
}
public class KK {
    public void gr(int k) {
        if (k==0) throw new Izuz();
        // regularno ponasanje metoda
    }
}
public class TestIzuz {
    public static void main(String args[]) {
        KK k=new KK();
        k.gr(0);
        System.out.println("Regularan kraj programa!");
    }
}
```

U ovom primeru bačeni izuzetak nije obrađen, pa će program završti sa radom pre ispisa `Regularan kraj programa!`

PROVERAVANI IZUZETAK - PRIMER

```
public class Izuz extends Exception{
    ...
}
public class KK {
    public void gr(int k) {
        if (k==0) throw new Izuz();
        // regularno ponasanje metoda
    }
}
public class TestIzuz {
    public static void main(String args[]) {
        KK k=new KK();
        k.gr(0);
        System.out.println("Regularan kraj programa!");
    }
}
```

Dve greške koje registruje kompajler

PROVERAVANI IZUZETAK - PRIMER

```
public class Izuz extends Exception{
    ...
}
public class KK {
    public void gr(int k) throws Izuz{
        if (k==0) throw new Izuz();
        // regularno ponasanje metoda
    }
}
public class TestIzuz {
    public static void main(String args[]) {
        KK k=new KK();
        try {
        k.gr(0);
        }
        catch(Izuz o) {
            // obrada situacije u kojoj k.gr() nije izvrsen na ocekivan nacin
        }
        System.out.println("Regularan kraj programa!");
    }}
```

Obavezan throws

Obavezan try/catch

U ovom primeru bačeni izuzetak je uhvaćen, pa će program nastaviti nakon catch-a dalje i ispisati
Regularan kraj programa!

PROVERAVANI IZUZECI - DODATAK

Java je **striktna u forsiranju proverenih izuzetaka**. Ako se pozove metod koji navodi izuzetak u njegovoj `throws` klauzuli postoje 3 mogućnosti:

- uhvatiti i obraditi izuzetak

```
try {
    // Code that originates an arithmetic exception
}
catch (ArithmeticException e) {
  // Deal with the exception here
}
```

- uhvaćeni izuzetak proslediti dalje (na pozivajući nivo)

```
try {
    // Code that originates an arithmetic exception
}
catch (ArithmeticException e) {
  // Deal with the exception here
  throw e;   // Rethrow the exception to the calling program
}
```

- uhvatiti izuzetak, generisati sopstveni i baciti ga pozivaocu

```
try {
    // Code that originates an arithmetic exception
}
catch (ArithmeticException e) {
  // Deal with the exception here
  throw new SomeNewEcepton();
         // Rethrow the exception
}
```

NAPOMENA

- Legalno je baciti izuzetke koji su izvedeni iz izuzetaka navedenih u throws klauzuli.

```
public class NulaArgument extends Izuz{
 ...
}
public class KK {
 public void gr(int k) throws Izuz{
     if (k==0) throw new NulaArgument();
     // regularno ponasanje metoda
 }
}
```

OBRADA IZUZETAKA I VIŠESTRUKU CATCH BLOKOVI

- U delu `try/catch` bloka

```
catch (ExceptionType1 identifier) {
        // …
}
```

kod koji obrađuje izuzetak se poziva za `ExceptionType1` ili bilo koju njegovu podklasu.

- Ako je u nekom `try/catch` bloku navedeno nekoliko `catch` blokova sa nekoliko tipova izuzetaka u istoj klasnoj hijearhiji, potrebno je blokove postaviti tako da se prvo hvata izuzetak najniže podklase, pa redom prema najvišoj superklasi.

```
// neispravna sekvenca catch blokova
// neće se prevesti
try {
      // try block code
} catch(Exception e){ ... }
  catch(ArithmeticException e){ ... }
```

EXCEPTION OBJEKTI

Klasa Throwable je bazna klasa za sve Java izuzetke i ima:

podatke

- Poruku (message) (koja se inicijalizuje u konstruktoru)
- Zapis o call steku (execution stack) u trenutku kada je izuzetak kreiran (pun naziv svih pozvanih metoda, plus broj linije u kojoj se poziv dogodio)

dva public konstruktora

- default kontruktor
- kontruktor koji prihvata argument tipa String (tu može biti smeštena informacija o prirodi greške)

public metode koje omogućavaju pristup zapisu poruka i steka:

- getMessage() - Vraća sadržaj poruke (null za većinu predefinisanih klasa)
- printStackTrace() - Ispis poruka i steka na standardni izlaz
- printStackTrace(PrintStream s)
- fillInStackTrace() - ažurira zapis steka za mesto gdje se poziv iste obavlja; koristi se kada se ponovo baca izuzetak da bi se ažurirao zapis steka kada je ponovo bačena (korisno kod bacanja sopstvenih izuzetaka)
  - e.fillInStackTrace();
  - throw e;

DEFINISANJE NOVIH IZUZETAKA

- Dva osnovna razloga:
  - Dodavanje informacije kada se dogodi neki standardni izuzetak
  - Greška koja se događa u vašem kodu zaslužna novog tipa izuzetka
- Kako?

```
// exception class – minimalna definicija
public class DreadfulProblemException extends Exception
    {
  // Konstruktori
    public DreadfulProblemException(){  }  // Default constructor
    public DreadfulProblemException(String s)  {
           super(s); // Call the base class constructor
    }
}
```

Što još dodati?

- druge konstruktore
- varijable instance za čuvanje dodatnih informacija
- pristupne metode za varijable instance s dodatnim informacijama

FINALLY BLOK

- Koristi se za pospremanje (clean-up)
- Asociran s određenim try blokom (kao i catch blok)
- Može se koristiti i s try blokom koji sadrži kod koji ne baca nikakav izuzetak:
  - za kod s višestrukim break ili return elementima,
  - vrednosti koje vratimo s return u finally bloku će pregaziti bilo koji return izvršen u try bloku.

```
int metod(){
     try {
          //...
           return 1;
     }
     finally {
          return 2;
     }
     // nedohvatljiv deo koda, kompajler ne bi dozvolio
}
```

Metod iz primera vraća 2.