

Izuzeci

Objektno-orjentisano programiranje 1



Izuzeci

- Izuzeci su objekti kojima se signalizira pojava neke greške prilikom izvršavanja programa
- Izuzetke je moguće
 - **generisati** – signaliziramo pojavu greške
 - **obraditi** – obrađujemo nastalu grešku
 - **proslediti** – metod prosleđuje izuzetak metodi koja ju je pozvala
- Postoje dve vrste izuzetaka
 - **proveravani (eng. *checked*)** – izuzetak koji se **mora** ili obraditi ili proslediti
 - **neproveravani (eng. *unchecked*)** – izuzetak koji se **ne mora** niti obraditi niti proslediti
 - Neproveravani izuzetak je u većini slučajeva prouzrokovao **semantičkom greškom (*bag*)** koja ne zavisi od spoljnih faktora
 - **Popravljamo grešku umesto obrade izuzetka**

Izuzeci

- Izuzeci su objekti klase **Exception** ili klasa koji nasleđuju klasu **Exception** (iz paketa **java.lang**)
- Neproveravani izuzeci su objekti klase **RuntimeException** (paket **java.lang**) koja direktno nasleđuje klasu **Exception**
- Nove tipove izuzetaka definišemo tako što nasledimo klasu **Exception** (bilo direktno bilo indirektno)
 - ako nasledimo **RuntimeException** onda definišemo novi tip neproveravanog izuzetka
- Neki neproveravani izuzeci iz paketa **java.lang**
 - NullPointerException, ArithmeticException, ClassCastException, IllegalArgumentException, NegativeArraySizeException, NoSuchElementException, ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException, NumberFormatException

```

public class Recnik {
    // mapa koja neku rec slika u sve njene prevode
    private HashMap<String, LinkedList<String>> r =
        new HashMap<String, LinkedList<String>>();

    public void dodajSaGreskom(String orig, String prevod) {
        LinkedList<String> prevodi = r.get(orig);
        prevodi.add(prevod); // null pointer exception!
    }

    public void dodaj(String orig, String prevod) {
        LinkedList<String> prevodi = r.get(orig);
        if (prevodi == null) {
            prevodi = new LinkedList<String>();
            r.put(orig, prevodi);
        }
        prevodi.add(prevod);
    }

    public static void main(String[] args) {
        Recnik r = new Recnik();
        r.dodajSaGreskom("programiranje", "programming");
    }
}

```

Exception in thread "main" [java.lang.NullPointerException](#)
 at Recnik.dodajSaGreskom([Recnik.java:13](#))
 at Recnik.main([Recnik.java:28](#))

Nepraveravani izuzeci

```
// ArithmeticException
```

```
int x = 5, y = 0, z = x / y;
```

```
// ClassCastException
```

```
Object o = new Integer(5);
```

```
String s = (String) o;
```

```
// NegativeArraySizeException
```

```
int[] niz1 = new int[-45];
```

```
// ArrayIndexOutOfBoundsException
```

```
int[] niz2 = new int[10];
```

```
niz2[42] = 24;
```

```
// StringIndexOutOfBoundsException
```

```
String s = "Aca";
```

```
char c = s.charAt(5);
```

```
// NumberFormatException
```

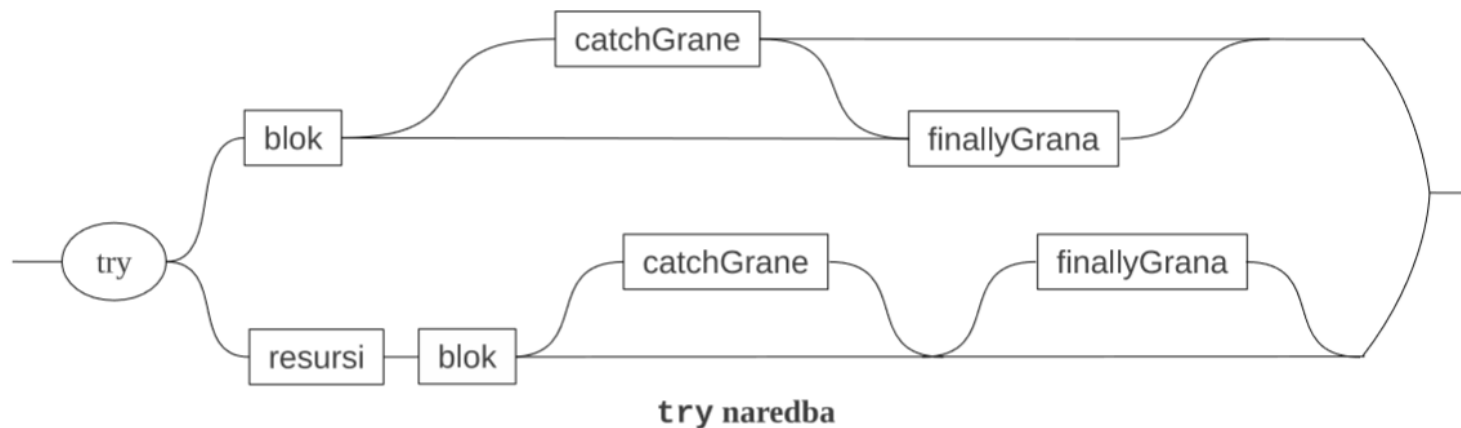
```
int x = Integer.parseInt("Mika");
```

- Program prekida sa radom i ispisuje se **stack trace** ako se
 - proveravani izuzetak prosleđuje sve do JVM (**main** metod prosledi neki proveravan izuzetak)
 - neproveravani izuzetak generiše i ne obradi (**a ne treba obrađivati ako su to faktički bagovi**)

```
java.lang.RuntimeException
  at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
  at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:39)
  at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:27)
  at java.lang.reflect.Constructor.newInstance(Constructor.java:513)
  at org.codehaus.groovy.reflection.CachedConstructor.invoke(CachedConstructor.java:77)
  at org.codehaus.groovy.runtime.callsite.ConstructorSite$ConstructorSiteNoUnwrapNoCoerce.callConstructor(ConstructorSite
  at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallConstructor(CallSiteArray.java:52)
  at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:192)
  at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:196)
  at newifyTransform$_run_closure1.doCall(newifyTransform.gdsl:21)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
  at java.lang.reflect.Method.invoke(Method.java:597)
  at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:86)
  at groovy.lang.MetaMethod.doMethodInvoke(MetaMethod.java:234)
  at org.codehaus.groovy.runtime.metaclass.ClosureMetaClass.invokeMethod(ClosureMetaClass.java:272)
  at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:893)
  at org.codehaus.groovy.runtime.callsite.PogoMetaClassSite.callCurrent(PogoMetaClassSite.java:66)
  at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callCurrent(AbstractCallSite.java:151)
  at newifyTransform$_run_closure1.doCall(newifyTransform.gdsl)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
  at java.lang.reflect.Method.invoke(Method.java:597)
  at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:86)
  at groovy.lang.MetaMethod.doMethodInvoke(MetaMethod.java:234)
  at org.codehaus.groovy.runtime.metaclass.ClosureMetaClass.invokeMethod(ClosureMetaClass.java:272)
  at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:893)
  at org.codehaus.groovy.runtime.callsite.PogoMetaClassSite.call(PogoMetaClassSite.java:39)
  at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.java:121)
  at org.jetbrains.plugins.groovy.dsl.GroovyDslExecutor$_processVariants_closure1.doCall(GroovyDslExecutor.groovy:54)
  at sun.reflect.GeneratedMethodAccessor61.invoke(Unknown Source)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
  at java.lang.reflect.Method.invoke(Method.java:597)
  at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:86)
  at groovy.lang.MetaMethod.doMethodInvoke(MetaMethod.java:234)
  at org.codehaus.groovy.runtime.metaclass.ClosureMetaClass.invokeMethod(ClosureMetaClass.java:272)
```

Obrada izuzetaka

- Izuzeci koji se mogu generisati u nekom bloku koda obrađuju **try-catch-finally** ili **try-with-resources** naredbom



- Ako se izuzetak generiše, kontrola toka se prebacuje na odgovarajuću **catch** granu
 - Kod za obradu grešaka nije izmešan sa kodom koji realizuje normalan tok izvršavanja → čitljiviji i razumljiviji kod
- Kod u **finally** grani se izvršava neovisno od toga da li je neki izuzetak bio generisan ili ne

Try-catch-finally naredba

```
PrintWriter pw = null;
try {
    pw = new PrintWriter(
        new BufferedWriter(new FileWriter("out.txt")));
    pw.println("Zdravo svete!");
} catch (IOException e) {
    System.out.println("Greska prilikom pisanja u fajl");
} finally {
    if (pw != null) pw.close();
}
```

- Možemo imati više *catch* grana
- *Finally* grana služi da oslobodimo zauzete resurse desila se greška ili ne
- *Finally* grana je obavezna ako nema nijedne *catch* grane
 - *Finally* bez *catch* – prosleđujemo izuzetak uz oslobađanje resursa

Try-catch-finally naredba

- **Možemo imati više *catch* grana**

```
PrintWriter pw = null;
try {
    pw = new PrintWriter(
        new BufferedWriter(new FileWriter("out.txt")));
    pw.println("Zdravo svete!");
} catch (FileNotFoundException e) {
    System.out.println("Greska prilikom kreiranja fajla");
} catch (IOException e) {
    System.out.println("Greska prilikom pisanja u fajl");
} finally {
    if (pw != null) pw.close();
}
```

- **Jedna *catch* grana može obrađivati više tipova izuzetaka (*multi-catch* grana)**

```
try {
    // pisemo nesto u bazu podataka
} catch (SQLException | IOException e) {
    System.out.println("Greska prilikom pisanja u bazu");
}
```

```

public class BrojacLinija {
    public static void main(String[] args) {
        if (args.length != 1) {
            S.o.p("Koriscenje: java BrojacLinija ImeUlaznogFajla");
            return;
        }

        File f = new File(args[0]);
        if (!f.exists() || !f.canRead()) {
            S.o.p("Ulazni fajl ne postoji ili se ne moze citati");
            return;
        }

        int brojLinija = 0;
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(f));
            while (br.readLine() != null) brojLinija++;
            S.o.p("Broj linija = " + brojLinija);
        } catch (IOException e) {
            System.out.println("Greska prilikom citanja fajla");
        } finally {
            try {
                if (br != null) br.close();
            } catch (IOException e) {
                S.o.p("Greska prilikom zatvaranja fajla");
            }
        }
    }
}

```

Finally blok

- Finally blok se takođe izvršava ako se izvršavanje metoda prekine return naredbom

```
public void pisiMozda() {  
    PrintWriter pw = null;  
    try {  
        pw = new PrintWriter(  
            new BufferedWriter(new FileWriter("o.txt")));  
  
        if (Math.random() < 0.5)  
            return; // izvrsava se finally blok  
  
        pw.println("Zdravo svete!");  
    } catch (IOException e) {  
        System.out.println("Greska");  
    } finally {  
        if (pw != null) pw.close();  
    }  
}
```

Try-with-resources

```
try (  
    PrintWriter pw = new PrintWriter(  
        new BufferedWriter(new FileWriter("out.txt")))  
    ) {  
        pw.println("Zdravo svete!");  
    } catch (IOException e) {  
        System.out.println("Greska prilikom pisanja u fajl");  
    }  
}
```

- Try naredba parametrizovana jednim ili više resursom
 - Resursi međusobno razdvojeni ;
 - Resurs je objekat klase koja implementira interfejs **Closeable** ili **AutoCloseable** (npr. sve klase koje realizuju tokove podataka)
- Na kraju try naredbe resursi bivaju automatski zatvoreni (poziva se **close()** metod)
- Try-with-resources ne mora imati ni *catch* ni *finally* grane

```

public class CopyTxtFile {
    public static void main(String[] args) {
        if (args.length != 2) {
            S.o.p("Pokretanje: java CopyTxtFile Src Dst");
            return;
        }

        File src = new File(args[0]);
        if (!src.exists() || !src.canRead()) {
            S.o.p("Izvorni fajl ne postoji ili se ne moze citati");
            return;
        }

        try (
            BufferedReader br =
                new BufferedReader(new FileReader(args[0]));
            PrintWriter pw =
                new PrintWriter(new BufferedWriter(
                    new FileWriter(args[1])))
        ) {
            String line;
            while ((line = br.readLine()) != null)
                pw.println(line);
        } catch (IOException e) {
            System.out.println("Greska prilikom kopiranja");
        }
    }
}

```

Prosleđivanje izuzetaka

- **Metoda može da prosledi izuzetak onoj metodi koja ju je pozvala**
- Za proveravane izuzetke je obavezno u zaglavlju metoda navesti da metod prosleđuje izuzetke
 - Iza ključne reči **throws** nabrajamo sve tipove izuzetaka koji se prosleđuju (razdvojene zarezima)

```
public class ZdravoSvete {  
    private static void pisi(String outFileName) throws IOException {  
        PrintWriter pw = new PrintWriter(  
            new BufferedWriter(new FileWriter(outFileName)));  
        pw.println("Zdravo svete");  
        pw.close();  
    }  
  
    public static void main(String[] args) {  
        try {  
            pisi("out.txt");  
        } catch (IOException e) {  
            System.out.println("Greska prilikom pisanja u fajl");  
        }  
    }  
}
```

Generisanje izuzetaka

- Izuzetke generišemo koristeći **throw** naredbu
- Ako metoda generiše proveravane izuzetke tada takve tipove izuzetaka treba specificirati u zaglavlju metode iza ključne reči **throws**

```
public String prvaDvaSlova(String rec) {  
    if (rec.length() < 2)  
        throw new IllegalArgumentException("Rec nema dva slova");  
  
    return rec.substring(0, 2);  
}  
  
public String prvaLinija(String ulazniFajl) throws IOException {  
    File f = new File(ulazniFajl);  
    if (!f.exists())  
        throw new IOException("Fajl ne postoji");  
    if (!f.canRead())  
        throw new IOException("Fajl ne moze da se cita");  
  
    try (BufferedReader br = new BufferedReader(new FileReader(f))) {  
        return br.readLine();  
    }  
}
```

Definisanje tipova izuzetaka

- Nove tipove izuzetaka definišemo nasleđujući klasu Exception direktno ili indirektno

```
public class KratakString extends Exception {  
    public KratakString() {  
        super("String ima manje od dva slova");  
    }  
}
```

```
public String prvaDvaSlova(String rec) throws KratakString {  
    if (rec.length() < 2) throw new KratakString();  
    return rec.substring(0, 2);  
}
```

```
public void stampajPrvaDva(String str) {  
    try {  
        System.out.println(prvaDvaSlova(str));  
    } catch (KratakString ks) {  
        System.out.println(ks.getMessage());  
    }  
}
```


Primer: Balansirane zagrade

```
if [<= n 1] {1} (* {n} (factorial (- n 1)))  
if  
  <= n 1  
  1  
  *  
    n  
    factorial  
    - n 1
```

Neke([]{ } zagrade)))))

Zatvorena]:20 bez otvorene

Ot (((nema zatvorene'

Otvorena (:5 nema zatvorenu

Zdravko{Zdr(av)ko[dren]}

Otvorena {:7 zatvorena sa]:23

```
public class Zagrada {  
    private int pozicija;  
    private char z;  
  
    public Zagrada(char z, int pozicija) {  
        this.z = z;  
        this.pozicija = pozicija;  
    }  
  
    public String toString() {  
        return z + ":" + pozicija;  
    }  
  
    public char getZagrada() {  
        return z;  
    }  
}
```

```
public class NepravilneZagrade extends Exception {  
    private Zagrada z;  
  
    public NepravilneZagrade(String poruka, Zagrada z) {  
        super(poruka);  
        this.z = z;  
    }  
  
    public Zagrada getZagrada() { return z; }  
}
```

```
public class OtvorenaViseca extends NepravilneZagrade {
    public OtvorenaViseca(Zagrada z) {
        super("Otvorena " + z + " nema zatvorenu", z);
    }
}

public class ZatvorenaViseca extends NepravilneZagrade {
    public ZatvorenaViseca(Zagrada z) {
        super("Zatvorena " + z + " bez otvorene", z);
    }
}

public class ZatvorenaPogresna extends NepravilneZagrade {
    public ZatvorenaPogresna(Zagrada otvorena, Zagrada zatvorena) {
        super("Otvorena " + otvorena + " zatvorena sa " + zatvorena, otvorena);
    }
}

private static boolean kompatibilna(char z1, char z2) {
    if (z1 == '{') return z2 == '}';
    else if (z1 == '[') return z2 == ']';
    else return z2 == ')';
}

private static StringBuilder uvuci(int level) {
    StringBuilder sb = new StringBuilder();
    sb.append('\n');
    for (int i = 0; i < level; i++) {
        sb.append(" ");
    }
    return sb;
}
```

```

public static String format(String ulaz) throws NepravilneZagrade {
    Stack<Zagrada> stek = new Stack<Zagrada>();
    StringBuilder sb = new StringBuilder();
    int level = 0;

    for (int i = 0; i < ulaz.length(); i++) {
        char c = ulaz.charAt(i);
        if (c == '(' || c == '[' || c == '{') {
            stek.push(new Zagrada(c, i));
            ++level;
            sb.append(uvuci(level));
        }
        else
            if (c == ')' || c == ']' || c == '}') {
                if (stek.empty())
                    throw new ZatvorenaViseca(new Zagrada(c, i));

                Zagrada poslednjaOtvorena = stek.pop();
                if (!kompatibilna(poslednjaOtvorena.getZagrada(), c)) {
                    throw new ZatvorenaPogresna(poslednjaOtvorena, new Zagrada(c, i));
                }

                level--;
            }
        else
            sb.append(c);
    }

    if (!stek.empty()) { throw new OtvorenaViseca(stek.pop()); }
    return sb.toString();
}

```

```

public static void main(String[] args) {
    String[] testSlucajevi = {
        "if [<= n 1] {1} (* {n} (factorial (- n 1)))",
        "Neke([]{ } zagrade ))))",
        "Ot (((  nema zatvorene",
        "Zdravko{Zdr(av)ko[dren]}"
    };

    int brZatvorenaPogresna = 0, brViseca = 0, ok = 0;
    for (int i = 0; i < testSlucajevi.length; i++) {
        try {
            System.out.println(PrettyPrinter.format(testSlucajevi[i]));
            ok++;
        } catch (NepravilneZagrade e) {
            System.err.println(e.getLocalizedMessage());

            if (e instanceof ZatvorenaPogresna)
                brZatvorenaPogresna++;
            else
                brViseca++;
        }
    }

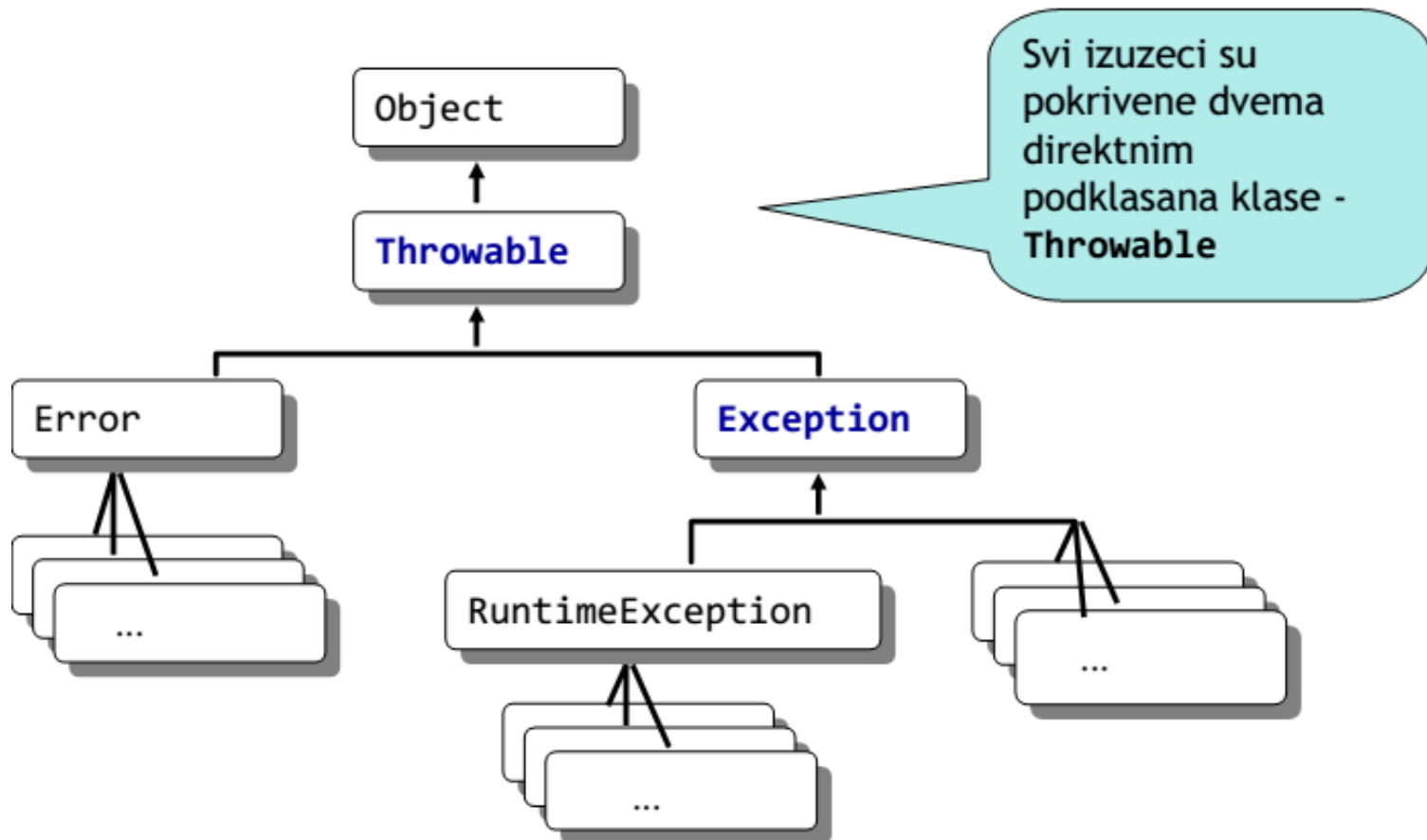
    System.out.println("Proslo test primera: " + ok);
    System.out.println("Zatvoreno pogresnih: " + brZatvorenaPogresna);
    System.out.println("Visecih          : " + brViseca);
}

```








Klasa Throwable

- Throwable

- Exception – izuzeci
- Error – izuzeci od kojih je oporavak uglavnom nemoguć, greške JVM



```
public class VMGreske {  
    private static void beskonacnaRekurzija() {  
        beskonacnaRekurzija();  
    }  
  
    public static void main(String[] args) {  
        try {  
            beskonacnaRekurzija();  
        } catch (StackOverflowError e) {  
            System.out.println("Prepunjen call stack");  
        }  
  
        try {  
            long[][][][] hiperKocka =  
                new long[Integer.MAX_VALUE][Integer.MAX_VALUE]  
                    [Integer.MAX_VALUE][Integer.MAX_VALUE];  
        } catch (OutOfMemoryError e) {  
            System.out.println("Malo memorije za hiperkocku. Povecaj xmx");  
        }  
  
        System.out.println("Nastavljam sa radom... ");  
    }  
}
```

 Problems  Javadoc  Declaration  Console  Call Hierarchy  History  Synchronizer

<terminated> VMGreske [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Dec 12, 2013 7:57:38 PM)

Prepunjen call stack

Malo memorije za hiperkocku. Povecaj xmx

Nastavljam sa radom...