

Iznimke u Javi

AUDITORNE VJEŽBE

Sadržaj

Ispis staze stoga iznimke

Zapisivanje podataka o iznimkama u log datoteke – LogBack

Apache Maven

Primjer zadatka s iznimkama – pogađanje brojeva

Pitanja s certifikata

Ispis staze stoga iznimke

- Staza stoga iznimke (engl. *stack trace*) sadržava ključne informacije o razlogu bacanja neke iznimke i često se ispisuje u konzolu razvojnog okruženja
- Ispis staze stoga u konzolu moguće je obaviti i pozivom metode „`printStackTrace`” iz objekta koji predstavlja iznimku, a najčešće se koristi unutar „`catch`” bloka
- Osim ispisa staze stoga, iz objekta koji predstavlja iznimku moguće je dohvatiti i poruku koja detaljnije opisuje razlog nastanka iznimke korištenjem metode „`getMessage`”:

```
catch(ArithmeticException ex1) {  
    ex1.printStackTrace();  
    String poruka = ex1.getMessage();  
    ...  
}
```

[java.lang.ArithmeticException: / by zero](#)
/ by zero

Zapisivanje podataka o iznimkama u log datoteke - LogBack

- Umjesto da se informacije o iznimkama zapisuju u konzolu koja ne sprema podatke, iste je moguće zapisivati u log datoteke
- Kako je to česta praksa radi omogućavanja naknadnog analiziranja uzroka problema u radu aplikacije, za to se koristi vanjska biblioteka LogBack
- Ona omogućava da se na jednostavan način konfiguriraju detalji koji se zapisuju u log datoteke i obavi samo zapisivanje
- Osim pogrešaka u log datoteke je moguće zapisivati i informativne poruke koje dokumentiraju aktivnosti korisnika u aplikaciji, sve u svrhu lakše rekonstrukcije i reproduciranja problema, a samim time i njihovog ispravljanja
- Biblioteka LogBack se konfigurira korištenjem XML datoteke koja se mora nalaziti unutar projekta koji koristi „logiranje”

Zapisivanje podataka o iznimkama u log datoteke - LogBack

- Primjer te XML datoteke izgleda ovako:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="FILE" class="ch.qos.logback.core.FileAppender">
    <file>logs/pogreske.log</file>
    <encoder>
      <pattern>%date %level [%thread] %logger{10} [%file:%line] %msg%n</pattern>
    </encoder>
  </appender>
  <root level="debug">
    <appender-ref ref="FILE" />
  </root>
</configuration>
```

Zapisivanje podataka o iznimkama u log datoteke - LogBack

- LogBack omogućava kreiranja nekoliko razina log zapisa u ovisnosti o njihovoj važnosti i detaljnosti:
 - ERROR
 - WARN
 - INFO
 - DEBUG
 - TRACE
- Java naredbe koje omogućavaju zapisivanje informacija u log datoteke ovise o „razini logiranja” što se očituje u nazivu metode koja se poziva, npr.:
`Logger.error("Došlo je do pogreške u radu aplikacije!", ex);`
- U metodu za „logiranje” se često predaje i objekt koji predstavlja iznimku

Zapisivanje podataka o iznimkama u log datoteke - LogBack

- Primjer sadržaja log datoteke:

```
2016-10-23 16:37:22,546 INFO [main] h.j.v.g.Glavna [Glavna.java:40] Započet unos podataka
```

```
2016-10-23 16:37:22,551 INFO [main] h.j.v.g.Glavna [Glavna.java:46] Započet unos 1. KLIJENTA:
```

```
2016-10-23 16:37:47,913 ERROR [main] h.j.v.g.Glavna [Glavna.java:271] Unesen neispravan format datuma: 12.12.2016....
```

```
java.time.format.DateTimeParseException: Text '12.12.2016....' could not be parsed, unparsed text found at index 11
```

```
at java.time.format.DateTimeFormatter.parseResolved0(DateTimeFormatter.java:1952)
```

```
at java.time.format.DateTimeFormatter.parse(DateTimeFormatter.java:1851)
```

```
at java.time.LocalDate.parse(LocalDate.java:400)
```

```
at hr.java.vjezbe.glavna.Glavna.unesiKlijenta(Glavna.java:268)
```

```
at hr.java.vjezbe.glavna.Glavna.main(Glavna.java:47)
```

Apache Maven

- Alat za pojednostavljenje upravljanja ovisnostima (engl. *dependencies*) o vanjskim bibliotekama
- Umjesto dodavanja kopija JAR datoteka koje predstavljaju vanjske biblioteke u sam Eclipse projekt, Apache Maven omogućava kreiranje lokalnog repozitorija koji se automatski ažurira resursima iz globalnog repozitorija na Internetu
- Ovisnostima se upravlja iz datoteke „pom.xml”
- Razvojno okruženje IntelliJ je opremljeno potrebnim dodacima za korištenje Mavena
- Da bi se projekt konfigurirao pomoću Mavena, na početku je potrebno pretvoriti ga u „Maven Project” korištenjem opcije „Add Framework Support -> Maven”

Apache Maven

- Potvrđivanjem konfiguracijskih parametara unutar projekta kreira se „pom.xml” datoteka sa sljedećim sadržajem:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>KlaseAutomobil</artifactId>
  <version>1.0-SNAPSHOT</version>

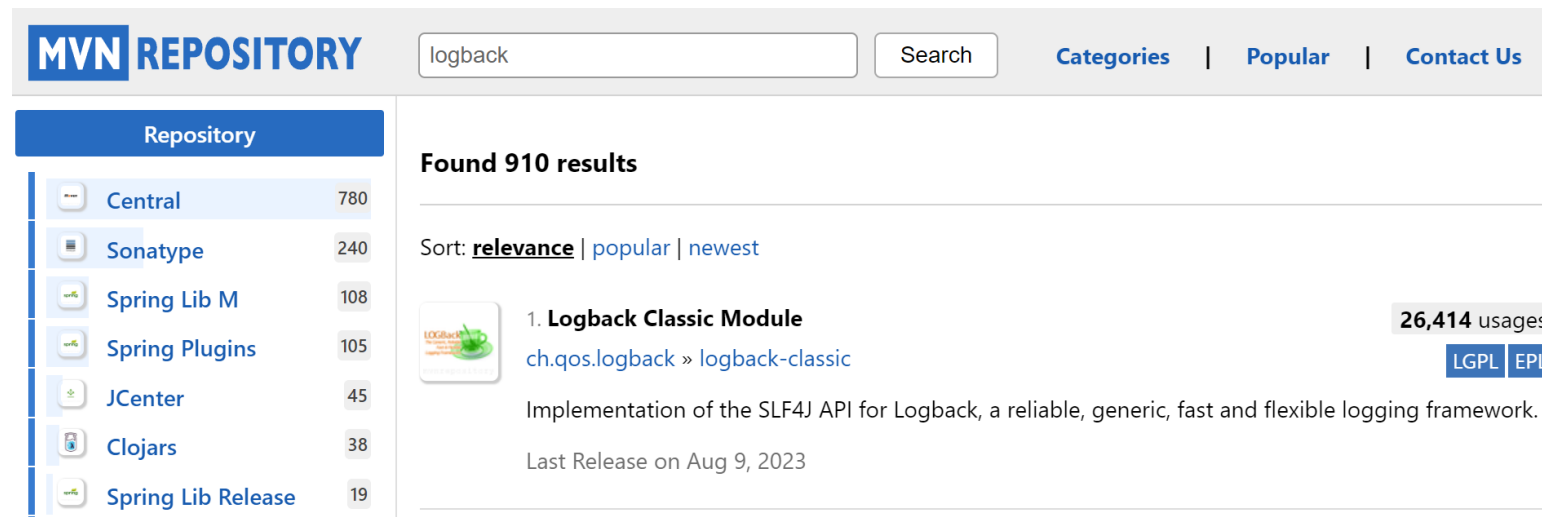
  <dependencies>
    <dependency>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
      <version>1.4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <properties>
    <maven.compiler.source>21</maven.compiler.source>
    <maven.compiler.target>21</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

</project>
```

Apache Maven

- Da bi se definirala vanjska biblioteka koju Maven mora dodati u „Build Path” projekta, prvo je potrebno u centralnom Maven repozitoriju pronaći odgovarajući dio XML konfiguracije koja se mora ubaciti u datoteku „pom.xml”
- To je moguće postići tako da se na stranici „<http://mvnrepository.com/>” u tražilicu upiše naziv biblioteke:



The screenshot shows the Maven Repository website. The header includes the 'MVN REPOSITORY' logo, a search bar with 'logback' entered, and links for 'Categories', 'Popular', and 'Contact Us'. A left sidebar lists various repositories with their respective artifact counts. The main content area displays 'Found 910 results' and sorting options. The top result is 'Logback Classic Module' with 26,414 usages, a description, and the last release date.

Repository	Count
Central	780
Sonatype	240
Spring Lib M	108
Spring Plugins	105
JCenter	45
Clojars	38
Spring Lib Release	19

Found 910 results

Sort: relevance | popular | newest

1. Logback Classic Module 26,414 usages


[ch.qos.logback](#) » [logback-classic](#) LGPL EPL

Implementation of the SLF4J API for Logback, a reliable, generic, fast and flexible logging framework.

Last Release on Aug 9, 2023

Apache Maven

- Nakon toga je potrebno odabrati verziju i dio konfiguracije koji je potrebno prebaciti u „pom.xml” koji je označen „tagovima” pod nazivom „dependency”:

**Logback Classic Module » 1.4.11**

Implementation of the SLF4J API for Logback, a reliable, generic, fast and flexible logging framework.

License	EPL 1.0 LGPL 2.1
Categories	Logging Frameworks
Tags	logback logging
Date	Aug 09, 2023
Files	pom (12 KB) jar (276 KB) View All
Repositories	Central
Ranking	#10 in MvnRepository (See Top Artifacts) #2 in Logging Frameworks
Used By	26,414 artifacts

[Maven](#) [Gradle](#) [Gradle \(Short\)](#) [Gradle \(Kotlin\)](#) [SBT](#) [Ivy](#) [Grape](#) [Leiningen](#) [Buildr](#)

```
<!-- https://mvnrepository.com/artifact/ch.qos.logback/logback-classic -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.4.11</version>
  <scope>test</scope>
</dependency>
```



Apache Maven

- XML konfiguraciju potrebno je ubaciti između „tagova” pod nazivom „dependencies” koje je potrebno smjestiti u „pom.xml” datoteku :

```
<dependencies>
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.4.11</version>
    </dependency>
</dependencies>
```

Primjer zadatka s iznimkama – pogađanje brojeva

```
public class PremaliBrojException extends Exception {  
  
    public PremaliBrojException(String poruka) {  
        super(poruka);  
    }  
  
    public PremaliBrojException(Throwable uzrok) {  
        super(uzrok);  
    }  
  
    public PremaliBrojException(String poruka, Throwable uzrok) {  
        super(poruka, uzrok);  
    }  
}
```

Primjer zadatka s iznimkama – pogađanje brojeva

```
public class PrevelikiBrojException extends Exception {  
  
    public PrevelikiBrojException(String poruka) {  
        super(poruka);  
    }  
  
    public PrevelikiBrojException(Throwable uzrok) {  
        super(uzrok);  
    }  
  
    public PrevelikiBrojException(String poruka, Throwable uzrok) {  
        super(poruka, uzrok);  
    }  
}
```

Primjer zadatka s iznimkama – pogađanje brojeva – Glavna.java (1)

```
package primjer.glavna;
```

```
import java.util.Random;
```

```
import java.util.Scanner;
```

```
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
```

```
import primjer.iznimke.PremaliBrojException;
```

```
import primjer.iznimke.PrevelikiBrojException;
```

```
public class Glavna {
```

```
    public static final int MAX_BROJ = 100;
```

```
    private static int generiraniBroj = 0;
```

```
    private static final Logger logger = LoggerFactory.getLogger(Glavna.class);
```

Primjer zadatka s iznimkama – pogađanje brojeva – Glavna.java (2)

```
public static void main(String[] args) {  
  
    Random dajBroj = new Random();  
  
    generiraniBroj = dajBroj.nextInt(MAX_BROJ);  
    Logger.info("Generirao si broj " + generiraniBroj);  
  
    Scanner unosBroja = new Scanner(System.in);  
  
    boolean pogodio = false;  
    int uneseniBroj = 0;
```


Primjer zadatka s iznimkama – pogađanje brojeva – Glavna.java (3)

```
do {
    System.out.println("Unesite broj");
    uneseniBroj = unosBroja.nextInt();
    logger.info("Unesen je broj " + uneseniBroj);

    try {
        provjera(uneseniBroj);
        pogodio = true;
    } catch (PremaliBrojException e) {
        logger.info(e.getMessage(), e);
        System.out.println(e.getMessage());
    } catch (PrevelikiBrojException ex) {
        logger.info(ex.getMessage(), ex);
        System.out.println(ex.getMessage());
    }
}while(pogodio == false);
```

Primjer zadatka s iznimkama – pogađanje brojeva – Glavna.java (4)

```
System.out.println("BRAVOOOO!!! Pogodili ste traženi broj: " + generiraniBroj);
unosBroja.close();
}

public static void provjera(int broj) throws PremaliBrojException, PrevelikiBrojException {
    if (broj < generiraniBroj) {
        throw new PremaliBrojException("Unijeli ste premali broj!!!");
    }
    else if (broj > generiraniBroj) {
        throw new PrevelikiBrojException("Unijeli ste preveliki broj!!!");
    }
}
}
```

Pitanja s certifikata (1)

- I. Which class has the fewest subclasses: `Exception`, `RuntimeException`, or `Error`? Select the correct statement.
 - A. The `Exception` class has fewer subclasses than the `RuntimeException` and `Error` classes.
 - B. The `RuntimeException` class has fewer subclasses than the `Exception` and `Error` classes.
 - C. The `Error` class has fewer subclasses than the `Exception` and `RuntimeException` classes.

Pitanja s certifikata (2)

Given:

```
public static void test() throws FileNotFoundException {  
    try {  
        throw FileNotFoundException();  
    } finally {  
    }  
}
```

Determine why it will not compile. Which statement is correct?

- A. The code will not compile without a catch clause.
- B. The code needs the new keyword after the throw keyword.
- C. The finally clause should be the final clause.
- D. There is no class called `FileNotFoundException`.

Pitanja s certifikata (3)

What new features came with Java 7 to enhance exception handling capabilities? (Choose all that apply.)

- A. The multi-catch feature
- B. The boomerang feature
- C. The try-with-resources feature
- D. The try-with-riches feature

Pitanja s certifikata (4)

Given:

```
String typeOfDog = "Mini Australian Shepherd";  
typeOfDog = null;  
System.out.println(typeOfDog.length);
```

Which of the following is true?

- A. A `NullPointerException` will be thrown.
- B. A `RuntimeException` will be thrown.
- C. An `IllegalArgumentException` will be thrown.
- D. A compilation error will occur.

Pitanja s certifikata (5)

```
class Emu {  
    static String s = "-";  
    public static void main(String[] args) {  
        try {  
            throw new Exception();  
        } catch (Exception e) {  
            try {  
                try { throw new Exception();  
                } catch (Exception ex) { s += "ic "; }  
                throw new Exception(); }  
            catch (Exception x) { s += "mc "; }  
            finally { s += "mf "; }  
        } finally { s += "of "; }  
        System.out.println(s);  
    } }  
}
```

- A. -ic of
- B. -mf of
- C. -mc mf
- D. -ic mf of
- E. -ic mc mf of
- F. -ic mc of mf
- G. Compilation fails

What is the result?

Pitanja s certifikata (6)

```
3. class SubException extends Exception { }
4. class SubSubException extends SubException { }
5.
6. public class CC { void doStuff() throws SubException { } }
7.
8. class CC2 extends CC { void doStuff() throws SubSubException { } }
9.
10. class CC3 extends CC { void doStuff() throws Exception { } }
11.
12. class CC4 extends CC { void doStuff(int x) throws Exception { } }
13.
14. class CC5 extends CC { void doStuff() { } }
```

What is the result? (Choose all that apply.)

- A. Compilation succeeds
- B. Compilation fails due to an error on line 8
- C. Compilation fails due to an error on line 10
- D. Compilation fails due to an error on line 12
- E. Compilation fails due to an error on line 14

Pitanja s certifikata (7)

```
3. class Infinity { }
4. public class Beyond extends Infinity {
5.     static Integer i;
6.     public static void main(String[] args) {
7.         int sw = (int)(Math.random() * 3);
8.         switch(sw) {
9.             case 0: { for(int x = 10; x > 5; x++)
10.                     if(x > 100000000) x = 10;
11.                     break; }
12.             case 1: { int y = 7 * i; break; }
13.             case 2: { Infinity inf = new Beyond();
14.                     Beyond b = (Beyond)inf; }
15.         }
16.     }
17. }
```

And given that line 7 will assign the value 0, 1, or 2 to `sw`, which are true?
(Choose all that apply.)

- A. Compilation fails
- B. A `ClassCastException` might be thrown
- C. A `StackOverflowError` might be thrown
- D. A `NullPointerException` might be thrown
- E. An `IllegalStateException` might be thrown
- F. The program might hang without ever completing
- G. The program will always complete without exception

Pitanja s certifikata (8)

```
3. public class OverAndOver {
4.     static String s = "";
5.     public static void main(String[] args) {
6.         try {
7.             s += "1";
8.             throw new Exception();
9.         } catch (Exception e) { s += "2";
10.        } finally { s += "3"; doStuff(); s += "4";
11.        }
12.        System.out.println(s);
13.    }
14.    static void doStuff() { int x = 0; int y = 7/x; }
15. }
```

What is the result?

- A. 12
- B. 13
- C. 123
- D. 1234
- E. Compilation fails
- F. 123 followed by an exception
- G. 1234 followed by an exception
- H. An exception is thrown with no other output

Given:

```
3. public class Gotcha {  
4.     public static void main(String[] args) {  
5.         // insert code here  
6.  
7.     }  
8.     void go() {  
9.         go();  
10.    }  
11. }
```

And given the following three code fragments:

```
I.     new Gotcha().go();  
II.    try { new Gotcha().go(); }  
        catch (Error e) { System.out.println("ouch"); }  
III.   try { new Gotcha().go(); }  
        catch (Exception e) { System.out.println("ouch"); }
```

When fragments I–III are added, independently, at line 5, which are true?

(Choose all that apply.)

- A. Some will not compile
- B. They will all compile
- C. All will complete normally
- D. None will complete normally
- E. Only one will complete normally
- F. Two of them will complete normally

Pitanja s certifikata (9)

Pitanja s certifikata (10)

```
2. class MyException extends Exception { }
3. class Tire {
4.     void doStuff() { }
5. }
6. public class Retread extends Tire {
7.     public static void main(String[] args) {
8.         new Retread().doStuff();
9.     }
10.    // insert code here
11.        System.out.println(7/0);
12.    }
13. }
```

And given the following four code fragments:

```
I.    void doStuff() {
II.   void doStuff() throws MyException {
III.  void doStuff() throws RuntimeException {
IV.   void doStuff() throws ArithmeticException {
```

When fragments I–IV are added, independently, at line 10, which are true? (Choose all that apply.)

- A. None will compile
- B. They will all compile
- C. Some, but not all, will compile
- D. All of those that compile will throw an exception at runtime
- E. None of those that compile will throw an exception at runtime
- F. Only some of those that compile will throw an exception at runtime

Pitanja?
