

Išimčių apdorojimas

2018 M.Karpinskas



Nuorodos

<https://docs.oracle.com/javase/tutorial/essential/exceptions/>

https://www.tutorialspoint.com/java/java_exceptions.htm

<https://www.codeguru.com/java/article.php/c18137/All-About-Exception-Handling-in-Java.htm>

<https://www.slideshare.net/ssuser8e9f37/exceptions-in-java-13584393>

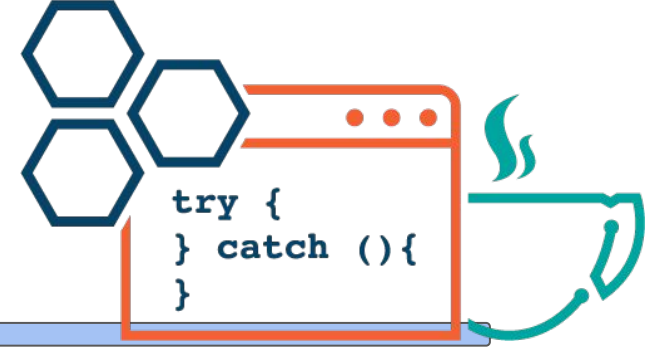
youtube.com

1. <https://www.youtube.com/watch?v=YCaIEDpu2oA>
2. https://www.youtube.com/watch?v=GbGExU_Mtsg
3. <https://www.youtube.com/watch?v=pV7AGogWHsk>
4. <https://www.youtube.com/watch?v=nuYdT5q3Gxc>

Kalbèsim

- 1. throw**
- 2. try - catch**
- 3. throws**
- 4. finally**

Išimčių apdorojimo sąvoka



- Ten, kur įvyko klaida, gali būti neaišku, kaip ją spręsti. Vykdymas sustabdomas. Klaidos apdorojimas perkeliamas į aukštesnį lygį
- To paties tipo klaidų apdorojimą galima perkelti į vieną vietą. Klaidų apdorojimas atskiriamas nuo kito kodo (**try catch**)

Išimtis vs. įprasta problema

Kuo skiriasi išimtinė situacija nuo įprastos problemos?

- Išimtinė situacija - tai problema, neleidžianti tęsti metodo ar kodo bloko vykdymo. Problemai spręsti einamajame kontekste trūksta informacijos. Viskas, ką galima padaryti, - tai peršokti iš einamojo konteksto į aukštesnį lygį. Tai atliekama generuojant išimtį
 - **throw new “exception type”**
- Įprasta problema – kai einamajame kontekste yra pakankamai informacijos problemai išspręsti. Tokia problema sprendžiama vietoje.
 - PVZ: įvestas netinkamas duomo paprašome dar kartą įvesti

throw

Kas įvyksta sugeneravus išimtį

1. Sukuriamas išimties objektas **new**
2. Iki šiol vykdyta veiksmų seka stabdoma
3. Į išimties objektą įdedama nuoroda į einamąjį kontekstą
4. Įsijungia išimčių apdorojimo mechanizmas ir nuo šios vietos pradeda ieškoti atitinkamo išimties apdorotojo (angl. exception handler)

Pvz.:

```
if(t==null) {  
    throw new NullPointerException("Ooops");  
}
```




```

1 package propogation;
2 import java.io.FileNotFoundException;
3
4 /** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
5 public class ExceptionTest {
6     public static void main(String[] args)
7         throws FileNotFoundException {
8         method1();
9         System.out.println("after calling m()");
10    }
11
12    static void method1() throws FileNotFoundException{
13        method2();
14    }
15
16    static void method2() throws FileNotFoundException{
17        method3();
18    }
19
20    static void method3() throws FileNotFoundException{
21        throw new FileNotFoundException();
22    }
23
24 }
25

```

step 6
(propagate exception)

step 5
(propagate exception)

step 4
(propagate exception)

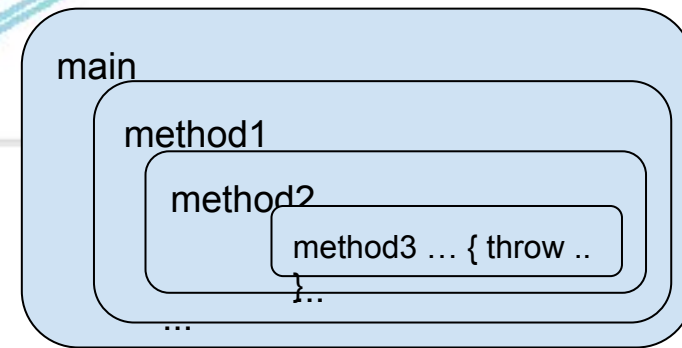
/*OUTPUT

```

Exception in thread "main" java.io.FileNotFoundException
    at propogation.ExceptionTest.method3(ExceptionTest.java:21)
    at propogation.ExceptionTest.method2(ExceptionTest.java:17)
    at propogation.ExceptionTest.method1(ExceptionTest.java:13)
    at propogation.ExceptionTest.main(ExceptionTest.java:8)

```

*/



main

method1

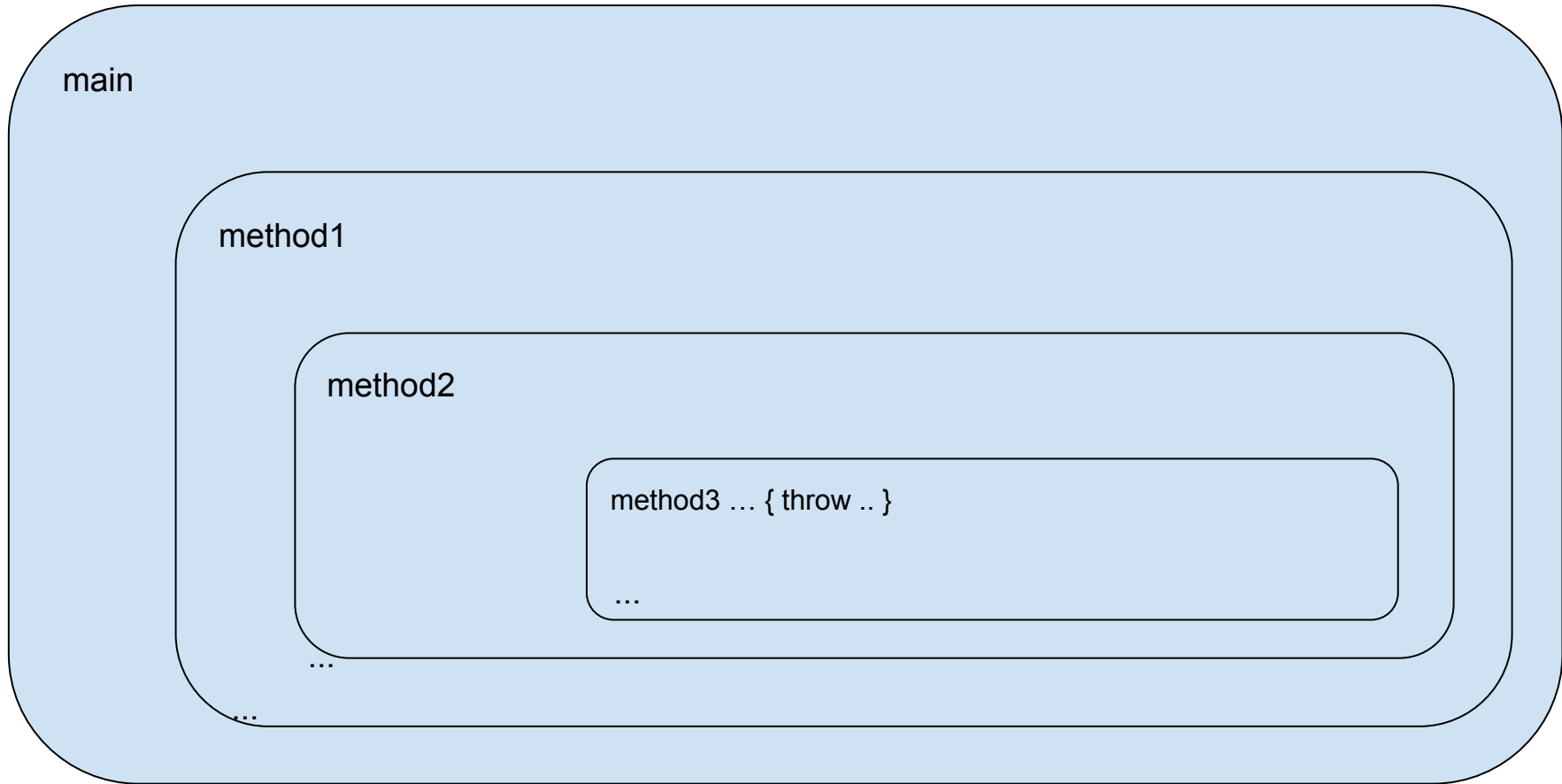
method2

method3 ... { throw .. }

...

...

...





```
1 package propogation;
2 import java.io.FileNotFoundException;
3
4 /** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
5 public class ExceptionTest {
6     public static void main(String[] args)
7         throws FileNotFoundException {
8         method1();
9         System.out.println("after calling m()");
10    }
11
12    static void method1() throws FileNotFoundException{
13        method2();
14    }
15
16    static void method2() throws FileNotFoundException{
17        method3();
18    }
19
20    static void method3() throws FileNotFoundException{
21        throw new FileNotFoundException();
22    }
23
24 }
25 }
```

step 1

step 2

step 3

step 4
(propagate exception)

step 5
(propagate exception)

step 6
(propagate exception)

/*OUTPUT

```
Exception in thread "main" java.io.FileNotFoundException
    at propogation.ExceptionTest.method3(ExceptionTest.java:21)
    at propogation.ExceptionTest.method2(ExceptionTest.java:17)
    at propogation.ExceptionTest.method1(ExceptionTest.java:13)
    at propogation.ExceptionTest.main(ExceptionTest.java:8)
```

*/

Pvz

```
package It.codeacademy.sdudy.exceptions;

public class FirstExc {
    public static void main(String[] args) {
        String tekstas = "Labas";
        printUpper(tekstas);
    }

    private static void printUpper(String tekstas) {
        System.out.println(tekstas.toUpperCase());
    }
}
```

SIMPLY EXPLAINED



`NullPointerException`

PVz

```
package It.codeacademy.sdudy.exceptions;
```

```
public class FirstExc {  
    public static void main(String[] args) {  
        String tekstas = null;  
        printUpper(tekstas);  
    }  
  
    private static void printUpper(String tekstas) {  
        System.out.println(tekstas.toUpperCase());  
    }  
}
```

```
Exception in thread "main" java.lang.NullPointerException  
    at It.codeacademy.sdudy.exceptions.FirstExc.printUpper(FirstExc.java:10)  
    at It.codeacademy.sdudy.exceptions.FirstExc.main(FirstExc.java:6)
```

Užd 1

Kaip patobulinti metodą, kad mestu klaidą su mūsų tekstiniu paaiškinimu?

```
private static void printUpper(String tekstas) {  
    System.out.println(tekstas.toUpperCase());  
}
```


Užd 1

Kaip patobulinti metodą, kad mestu klaidą su mūsų tekstiniu paaiškinimu?

```
private static void printUpper(String tekstas) {  
    System.out.println(tekstas.toUpperCase());  
}
```

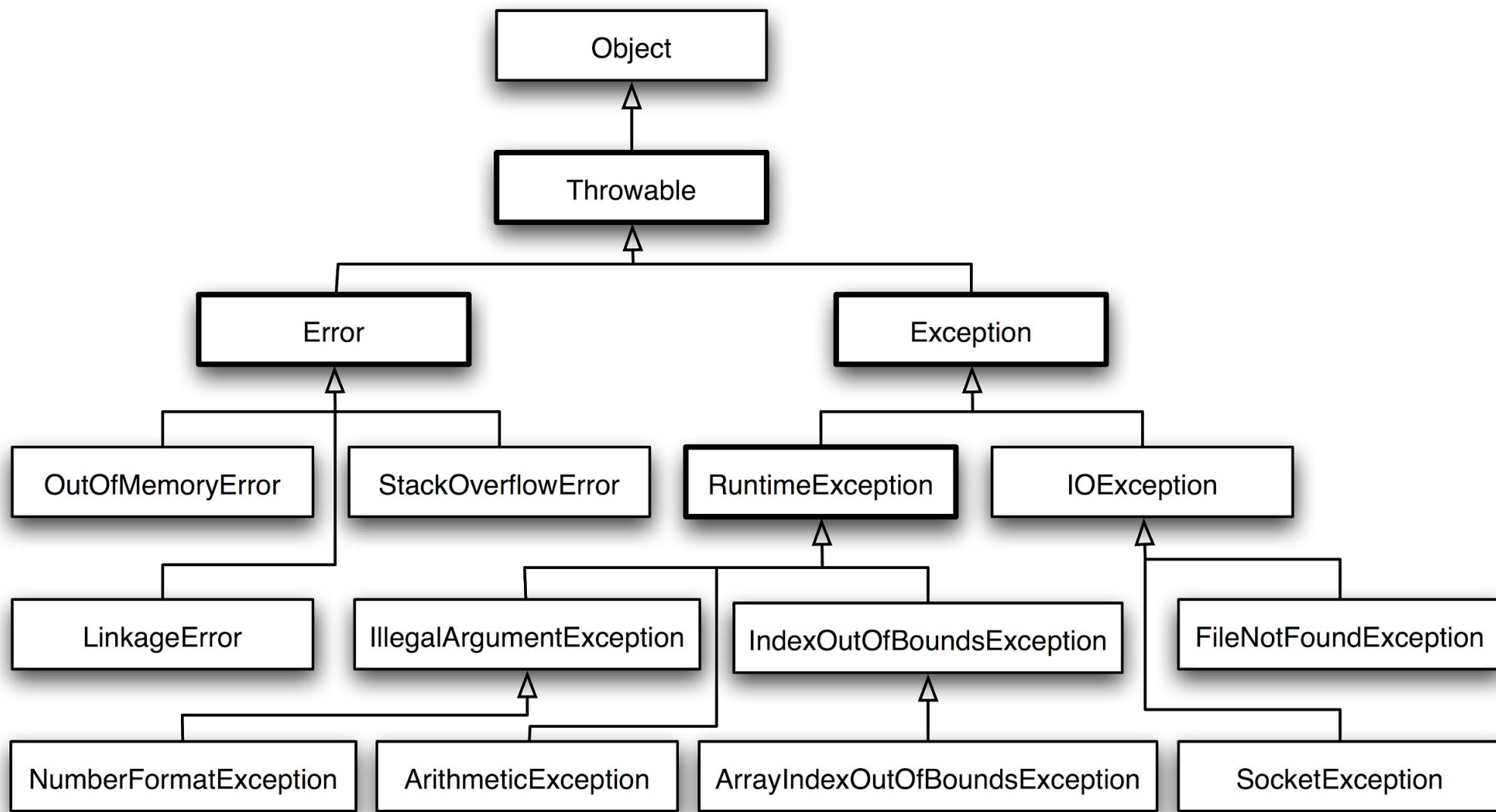
PVZ

```
if(t==null) {  
    throw new NullPointerException("0ooops");  
}
```


Pvz

```
package lt.codeacademy.sdudy.exceptions;
```

```
public class FirstExc {  
    public static void main(String[] args) {  
        String tekstas = null;  
        printUpper(tekstas);  
    }  
  
    private static void printUpper(String tekstas) {  
        if (tekstas == null) {  
            throw new IllegalArgumentException(  
                "Kintamasis tekstas turi buti ne null!!");  
        }  
        System.out.println(tekstas.toUpperCase());  
    }  
}
```



Išimties parametrai

- Išimties objektas kuriamas naudojant **new**, kuris išskiria atmintį ir iškviečia konstruktorių
- Yra N konstruktoriai: be parametrų ir su vienu String tipo parametru ir...
- Išimties objekto tėvinė klasė yra Throwable
- Informacija apie klaidą yra išimties objekte, be to, klaidos pobūdį nusako išimties objekto klasės vardas
- Apdorotas išimtis sunaikina šiukšlių surinkėjas

java.lang

Object

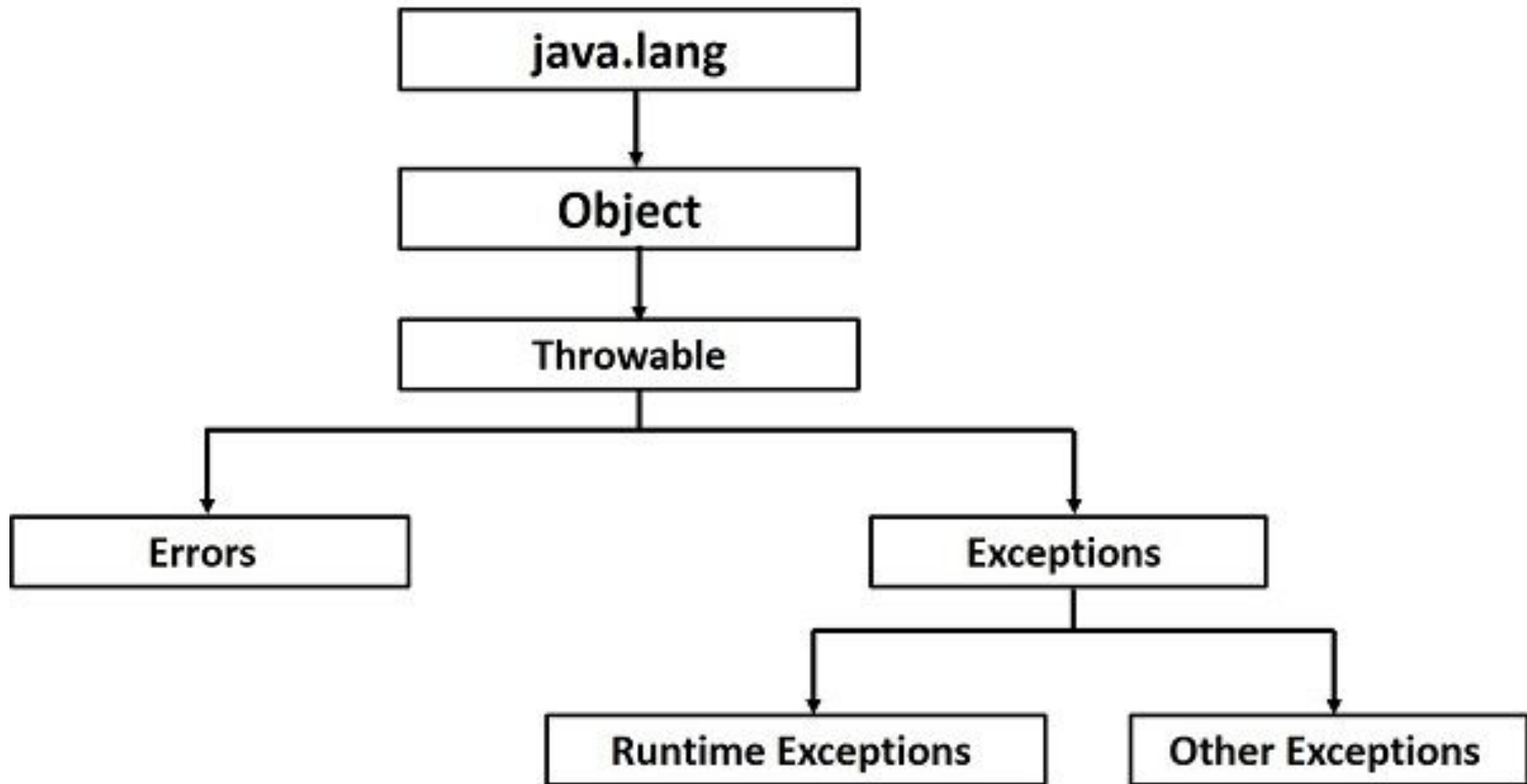
Throwable

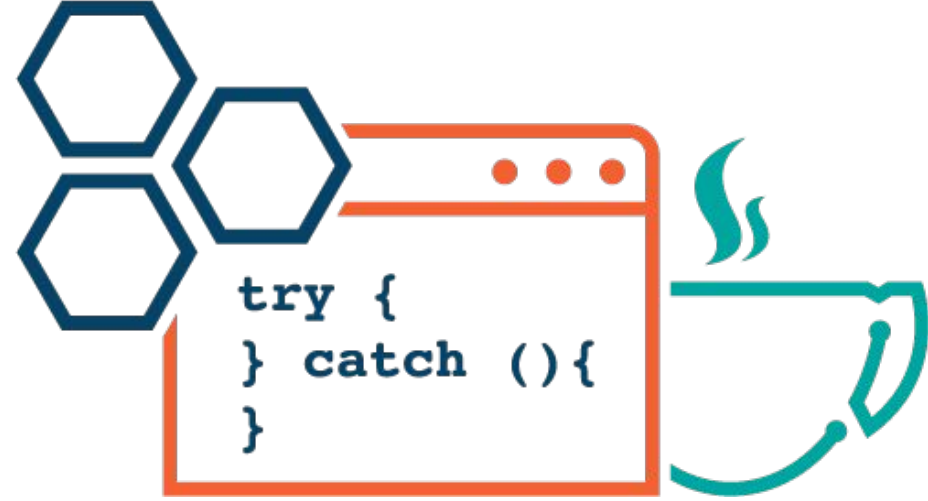
Errors

Exceptions

Runtime Exceptions

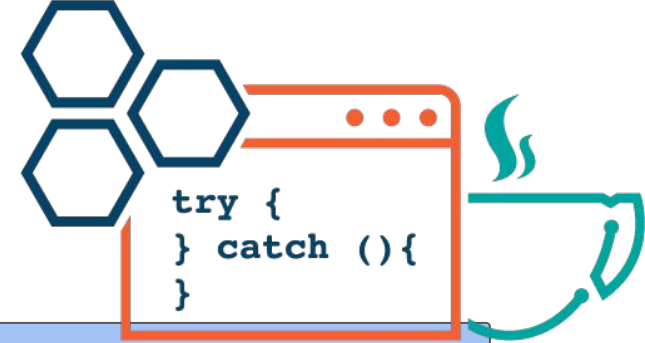
Other Exceptions





try - catch

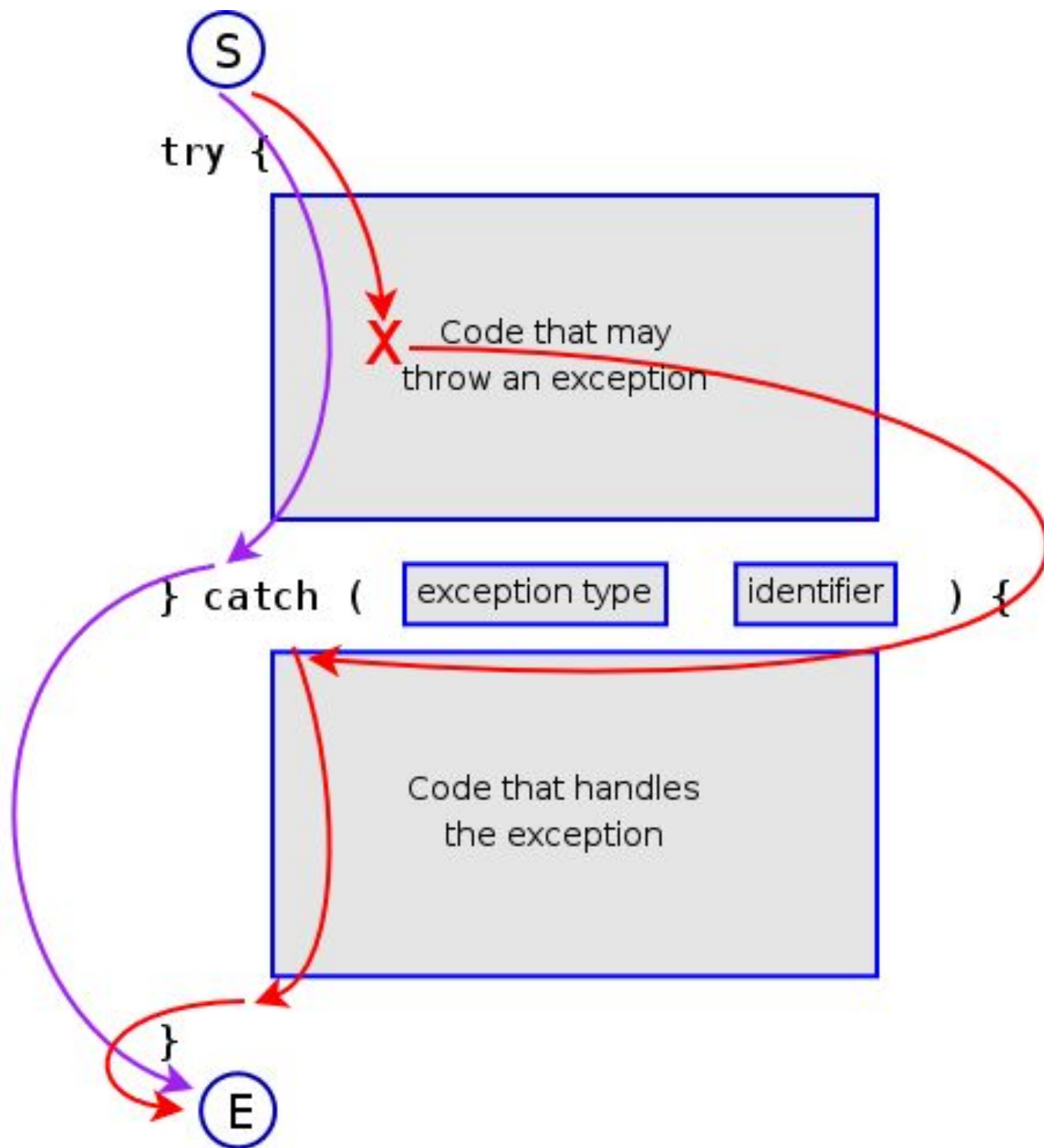
try blokas



```
try {  
  //kodas, galintis generuoti išimtis  
}
```

```
try {  
  //kodas, galintis generuoti išimtis  
} catch (Tipas1 id1) {  
  //išimčių, kurių tipas Tipas 1, apdorojimas  
} catch (Tipas2 id2) {  
  //išimčių, kurių tipas Tipas 2, apdorojimas  
}
```

ir t. t.



try blokas JAVA 8

```
try {  
    //kodas, galintis generuoti išimtis  
} catch (Tipas1 | Tipas2 e) {  
    //išimčių, kurių tipas Tipas 1 ir Tipas 2, apdorojimas  
} ir t. t.
```


PvzFirstExc

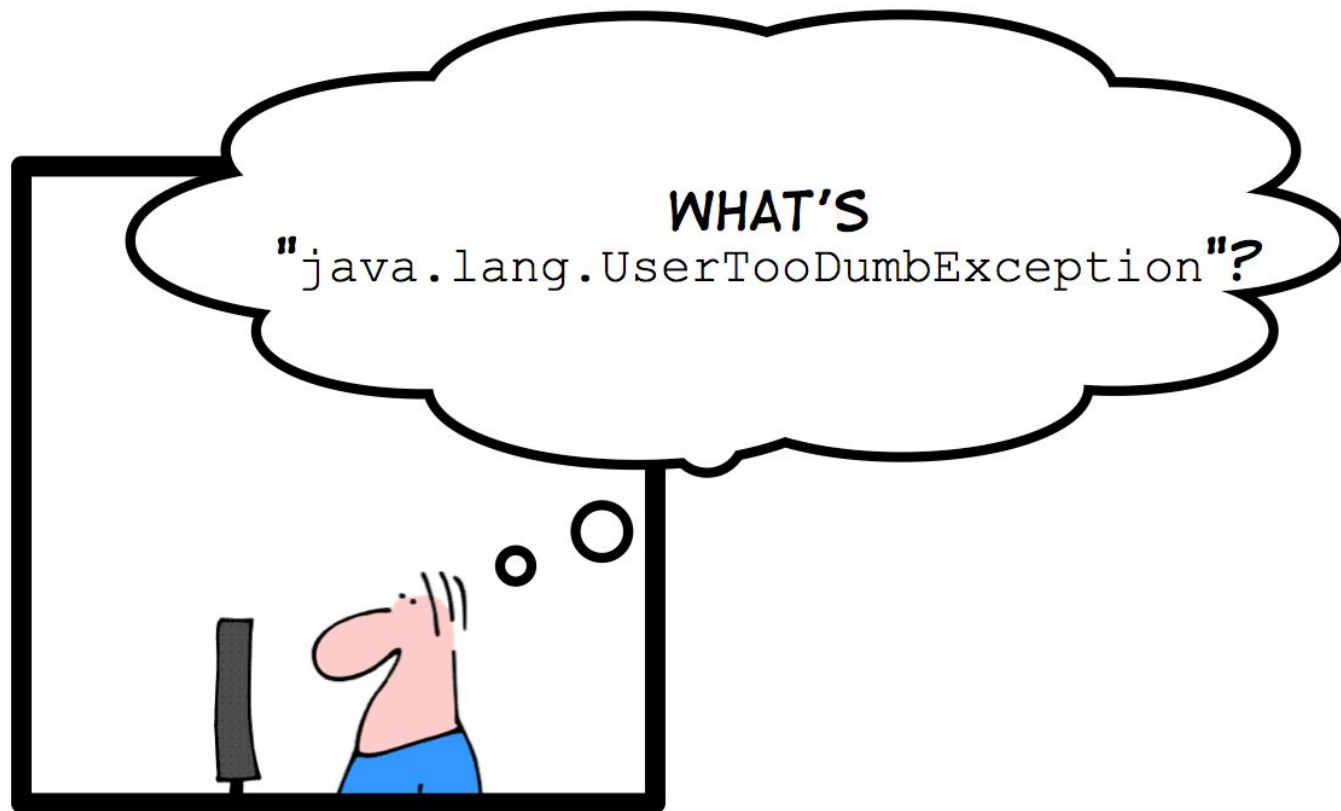
```
public class FirstExc {
    public static void main(String[] args) {
        String tekstas = "Vilnius";
        int i = 0;
        doMore(tekstas, i);
    }
    private static void doMore(String tekstas, int i) {
        try {
            System.out.println(tekstas.toUpperCase());
            System.out.println(100 / i);
        } catch (NullPointerException e) {
            e.printStackTrace();
        } catch (ArithmeticException e) {
            e.printStackTrace();
        }
        System.out.println("Pabaiga");
    }
}
```

Nuosavų išimčių kūrimas

```
public class MyException {
    public static void main(String[] args) {
        try {
            mesk();
        } catch (NullPointerException e) {
            System.out.println(1);
        } catch (SomeException e) {
            System.out.println(2);
        } catch (ArithmeticException e) {
            System.out.println(3);
        }
    }
    private static void mesk() {
        throw new SomeException();
    }
}
class SomeException extends RuntimeException {}
```

Try catch

geek & poke



WHAT'S
`"java.lang.UserTooDumbException"?`

CODER'S REVENGE

Užd 2

Patobulinkime metodą taip, kad mestų jūsų sukurta klaidos išimtį: BestException

```
private static void printUpper(String tekstas) {  
    System.out.println(tekstas.toUpperCase());  
}
```

throws

Užd 2

Patobulinkime metodą taip, kad mestų jūsų sukurtą klaidos išimtį: BestException

```
private static void printUpper(String tekstas) {  
    System.out.println(tekstas.toUpperCase());  
}
```

PVZ: **class** BestException **extends** RuntimeException {}

Išimčių specifikavimas

- Checked & Unchecked Exceptions
- **Unchecked Exceptions:**
 - paveldi iš RuntimeException ir Error
 - Specifikuoti nereikia: pvz., void f() ~~throws TooBig, TooSmall~~ {
- **Checked Exceptions:**
 - Kuriant metodą būtina nurodyti, kokias išimtis gali generuoti šis metodas, pvz., void f() **throws** TooBig, TooSmall {
 - Jei throws nėra (pvz., void f() {}), reiškia metodas išimčių (ne RuntimeException) generuoti negali.
 - Leidžiama specifikuoti **throws** daugiau išimčių, nei iš tikrųjų gali būti generuojama

java.lang

Object

Throwable

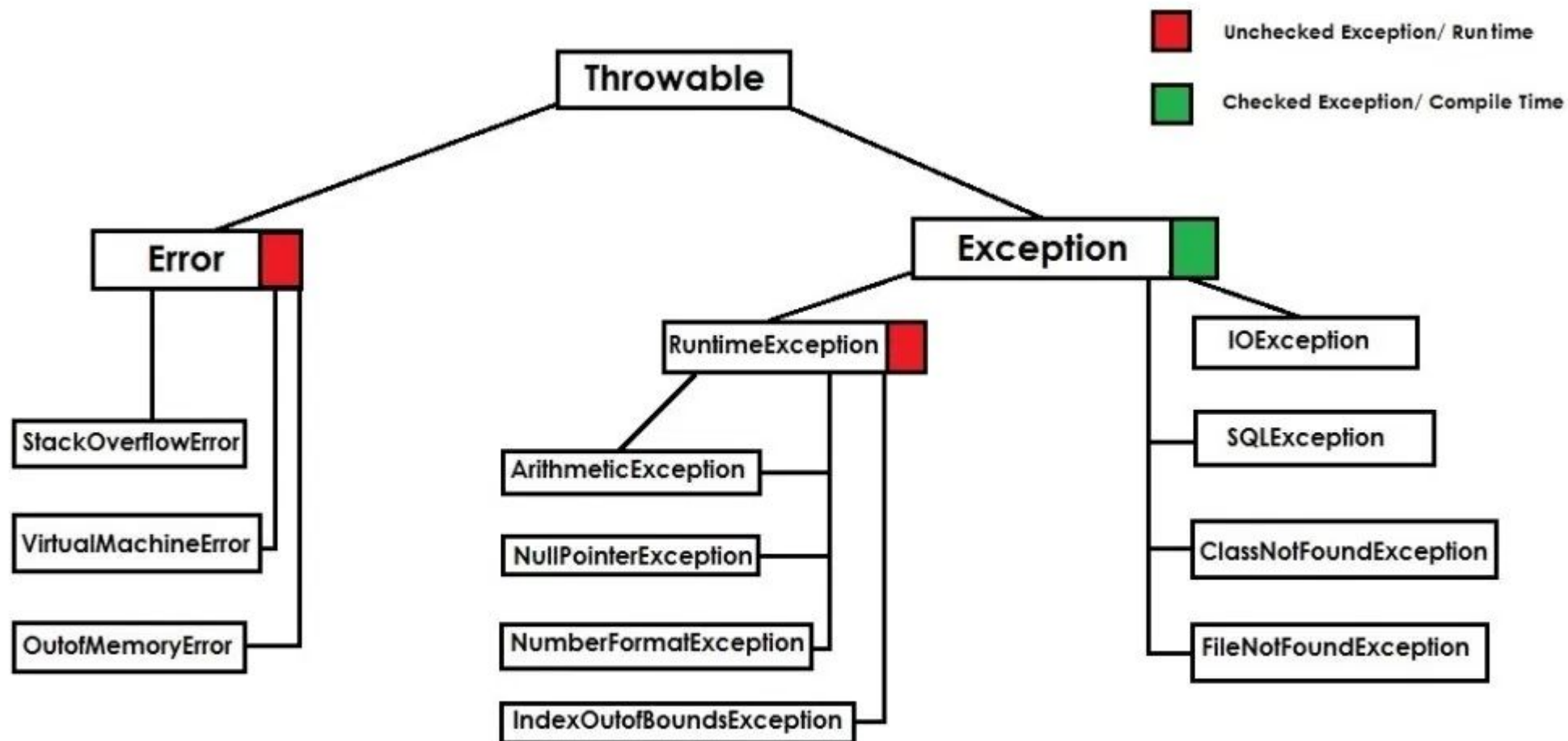
Errors

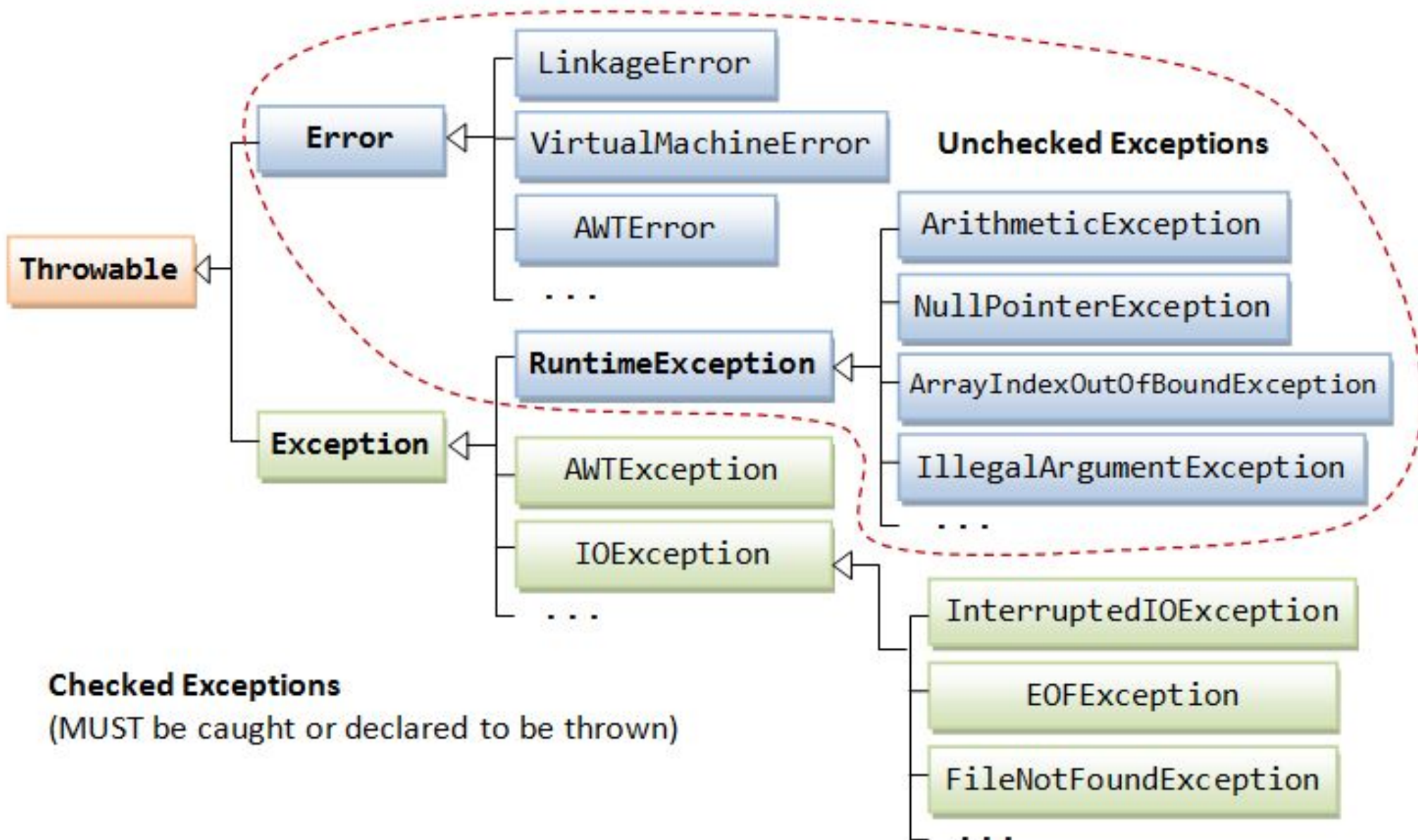
Exceptions

Runtime Exceptions

Other Exceptions







Links to read

<https://docs.oracle.com/javase/tutorial/essential/exceptions/>

https://www.tutorialspoint.com/java/java_exceptions.htm

Užd 3

Patobulinkime metodą taip, kad mestų jūsų sukurtą klaidos išimtį **Checked** tipo -> Exceptions - **extends** Exception

```
private static void printUpper(String tekstas) {  
    System.out.println(tekstas.toUpperCase());  
}
```

PVZ: **class** MyException **extends** Exception {}

```
} catch (
```

```
public class MyException {  
    public static void main(String[] args) {  
        try {  
            mesk();  
        } catch (NullPointerException e) {  
            System.out.println(1);  
        } catch (SomeException e) {  
            System.out.println(2);  
        } catch (ArithmeticException e) {  
            System.out.println(3);  
        } catch (CheckedException e) {  
            System.out.println(4);  
        }  
    }  
    private static void mesk() throws CheckedException {  
        throw new CheckedException();  
    }  
}  
class CheckedException extends Exception {}
```

Visų išimčių gaudymas

Gaudant bazinės klasės išimtis sugaunamos ir vaikinių klasių išimtys,

t. y. `catch(Exception e) {`

Šiuo atveju gaunama mažai informacijos apie konkrečią išimtį, tačiau ją galima gauti pasinaudojus bazinės klasės `Throwable` metodais

PVZ

```
class MyException2 {  
  
    public static void main(String[] args) {  
        try {  
            mesk();  
        } catch (Exception e) {  
            System.out.println(0);  
        }  
    }  
  
    private static void mesk() throws CheckedException {  
        throw new CheckedException();  
    }  
}  
  
class CheckedException extends Exception {  
  
}
```


Throwable metodai

- String getMessage()
- String getLocalizedMessage()
- String toString()
- void printStackTrace()
- Throwable fillInStackTrace()
- Object metodus getClass() ir klasės metodus getName()

printStackTrace() rezultato pavyzdys

lt.codeacademy.sdudy.exceptions.CheckedException

at lt.codeacademy.sdudy.exceptions.MyException2.mesk(MyException.java:40)

at lt.codeacademy.sdudy.exceptions.MyException2.mesk1(MyException.java:36)

at lt.codeacademy.sdudy.exceptions.MyException2.main(MyException.java:29)

Pakartotinis išimties generavimas

```
class MyException2 {  
    public static void main(String[] args) throws Exception {  
        try {  
            mesk();  
        } catch (Exception e) {  
            throw e;  
        }  
    }  
    private static void mesk() throws CheckedException {  
        throw new CheckedException();  
    }  
}  
class CheckedException extends Exception {}
```

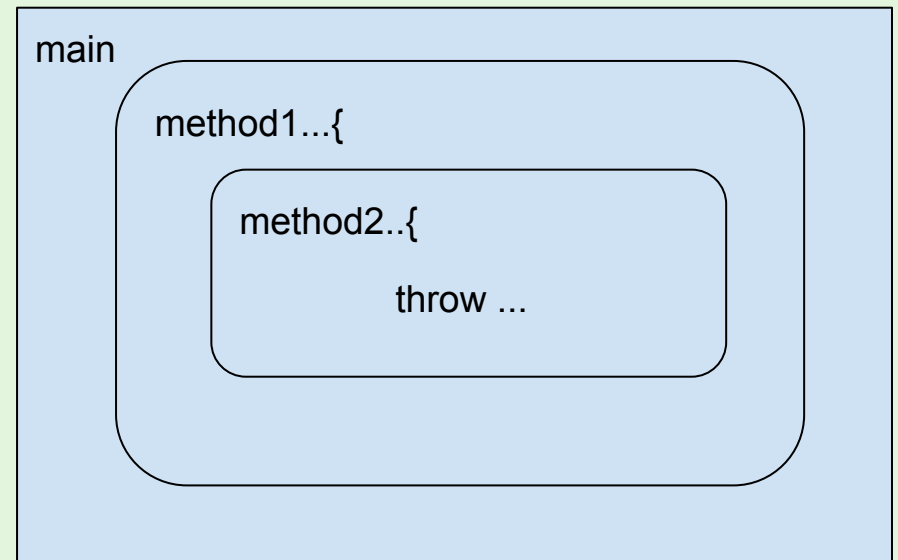
Exception in thread "main" [It.codeacademy.sdudy.exceptions.CheckedException](#)
at It.codeacademy.sdudy.exceptions.MyException2.mesk([MyException.java:36](#))
at It.codeacademy.sdudy.exceptions.MyException2.main([MyException.java:29](#))



Užd 4

- Sukurkime klasę su dviem metodais.
- Antrame metode metame exception (mūsų sukurta **RuntimeException**)
- Pirmame metode kviečiame antrą metodą ir pagauname exception'ą, išspausdiname informaciją ir metame į aukštesnį lygį

Main metode patestuoti, kaip viskas veikia

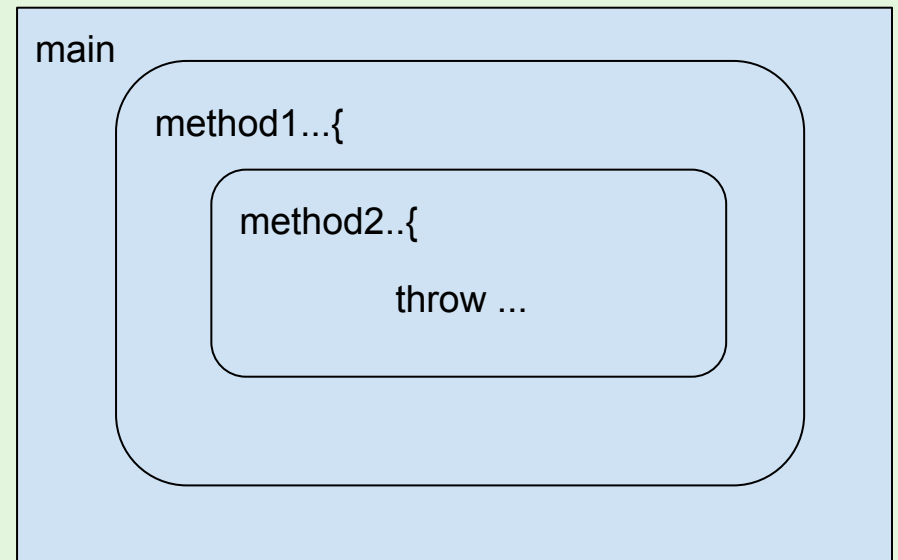


PVZ: **class** BestException **extends** RuntimeException {}

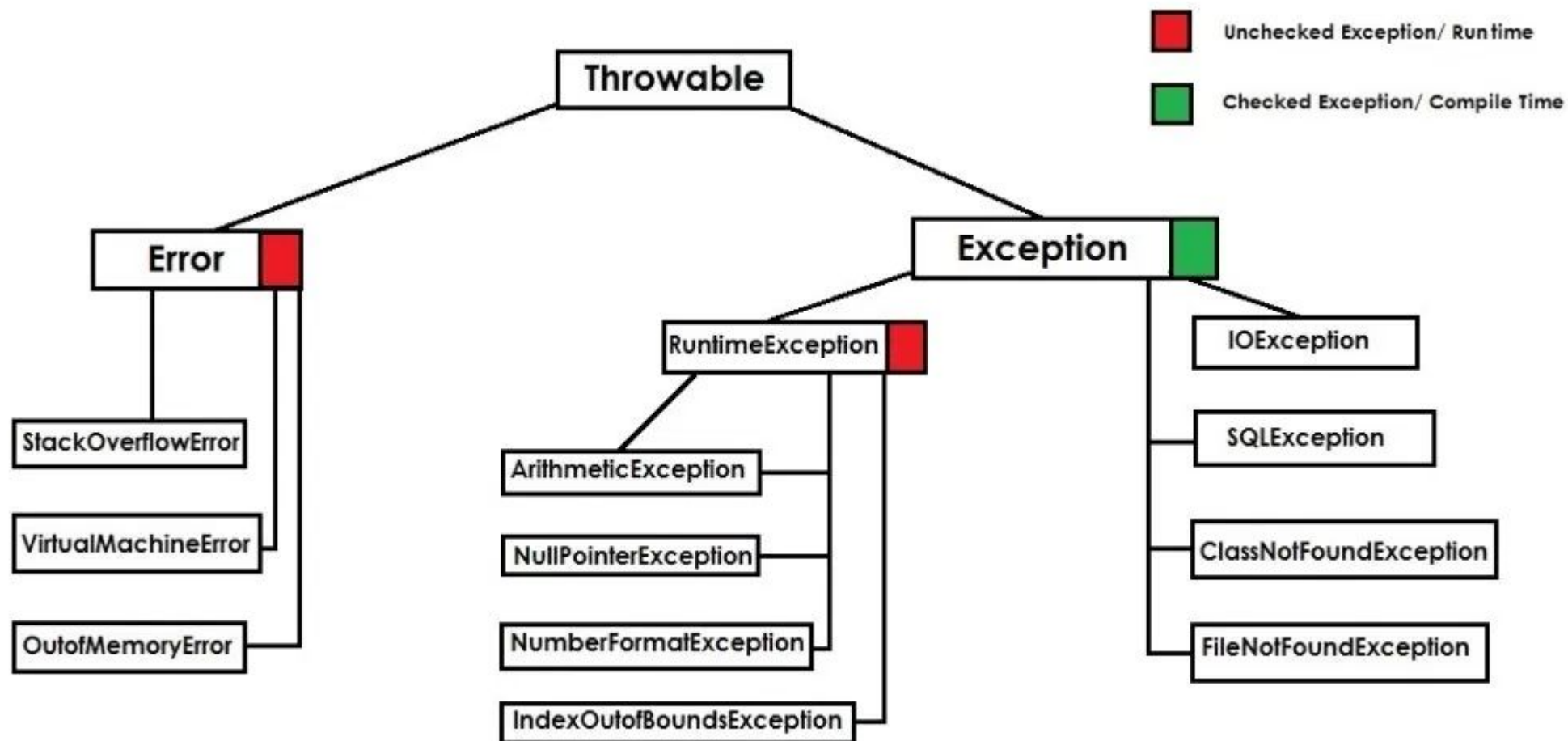
Užd 5

- Sukurkime klasę su dviem metodais.
- Antrame metode metame exception (mūsų sukurta **Exception**)
- Pirmame metode kviečiame antrą metodą ir pagauname exception'ą, išspausdiname informaciją ir metame į aukštesnį lygį

Main metode patestuoti, kaip viskas veikia



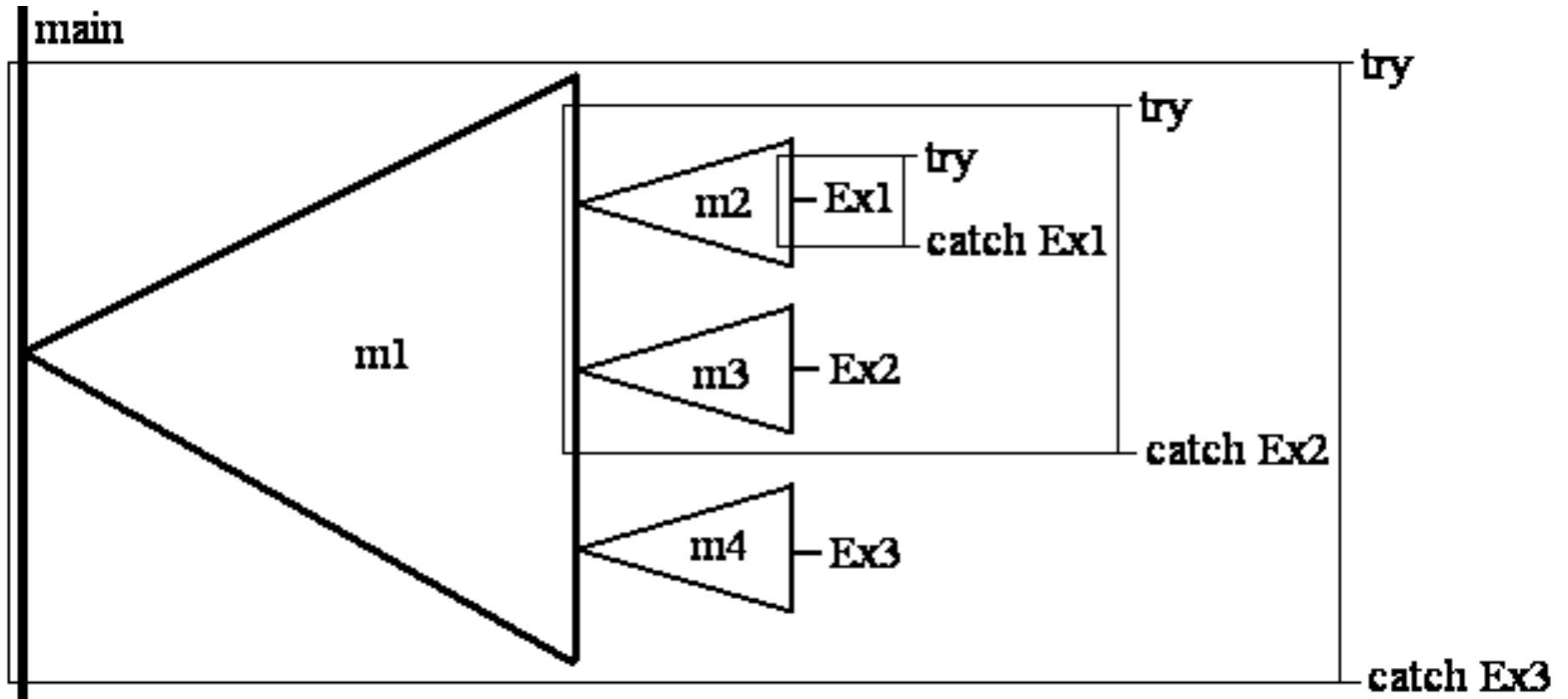
PVZ: **class** BestException **extends** Exception {}



Kai kurių java.lang išimčių vardai

- **ArrayIndexOutOfBoundsException**
- **ClassCastException**
- ClassNotFoundException
- IllegalArgumentException
- NegativeArraySizeException
- NoSuchFieldException
- **NullPointerException**
- **NumberFormatException**

Išimčių apdorojimas





finally

Sakinys finally

```
try {  
    //kodas, galintis generuoti išimtis  
} catch (Tipas1 id1) {  
    //išimčių, kurių tipas Tipas 1, apdorojimas  
} catch (Tipas2 id2) {  
    //išimčių, kurių tipas Tipas 2, apdorojimas  
} finally {  
    //veiksmai, kurie atliekami visada  
}
```

Kada naudoti finally?

- **Nenaudojamas** atminčiai atlaisvinti, tai atlieka šiukšlių surinkėjas
- **Naudojamas** failams uždaryti, tinklo ryšiams nutraukti, ekranui išvalyti ...
- finally vykdomas visada, net jei išimtis nebuvo generuota ar nebuvo sugauta
- Trūkumas: jei sakinyje finally generuosim naują išimtį, tai prieš tai buvusi bus prarasta

PvZ

```
class MyException3 {  
    public static void main(String[] args) throws Exception {  
        try {  
            mesk();  
        } catch (Exception e) {  
            throw e;  
        } finally {  
            throw new SomeException();  
        }  
    }  
    private static void mesk() throws CheckedException {  
        throw new CheckedException();  
    }  
}  
class CheckedException extends Exception {}  
class SomeException extends RuntimeException {}
```

Exception in thread "main" [It.codeacademy.sdudy.exceptions.SomeException](#)
at It.codeacademy.sdudy.exceptions.MyException3.main([MyException.java:49](#))

Užd 6

- Sukurkime klasę su dviem metodais.
- Antrame metode metame exception (mūsų sukurta **Exception**)
- Pirmame metode
 - kviečiame antrą metodą
 - ir pagauname exception'ą,
 - išspausdiname informaciją
 - ir metame į aukštesnį lygį
 - **final** bloke išspausdiname tekstą

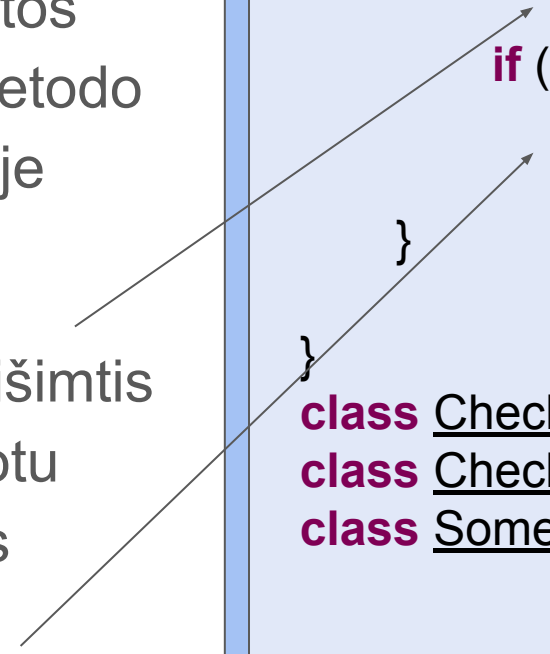
Main metode patestuoti, kaip viskas veikia

Apribojimai

Metodas gali generuoti tik tas išimtis, kurios buvo specifiкуotos (išvardintos) metodo bazinėje versijoje

1. arba iš jų paveldėtas išimtis iš specifiкуotu
2. Arba išimtys paveldi RuntimeException

```
private static void mesk() throws CheckedException {  
    int i = 0;  
    if (i == 0)  
        throw new CheckedException();  
    if (i == 0)  
        throw new CheckedException2();  
    if (i == 0)  
        throw new CheckedException();  
}  
}  
  
class CheckedException extends Exception {}  
class CheckedException2 extends CheckedException {}  
class SomeException extends RuntimeException {}
```



Apribojimai

Metodas gali generuoti tik tas išimtis, kurios buvo specifiкуotos (išvardintos) metodo bazinėje versijoje

1. arba iš jų paveldėtas išimtis iš specifiкуotu
2. Arba išimtys paveldi RuntimeException

```
private static void mesk() throws CheckedException,  
                                CheckedException2 {  
  
    int i = 0;  
    if (i == 0)  
        throw new CheckedException();  
    if (i == 0)  
        throw new CheckedException2();  
    if (i == 0)  
        throw new CheckedException();  
  
}  
  
}  
  
class CheckedException extends Exception {}  
class CheckedException2 extends Exception {}  
class SomeException extends RuntimeException {}
```


Apribojimai

1. Metodas gali specifiškai mažiau išimčių, negu bazinė metodo versija arba visai jų negeneruoti (gali gaudyti išimtis)

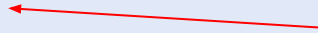
```
class A{  
    void m()throws CheckedException{  
  
    }  
}  
class B extends A{  
    @Override  
    void m() {  
  
    }  
}  
class CheckedException extends Exception {}  
class CheckedException2 extends Exception {}  
class SomeException extends RuntimeException {}
```

Apribojimai

1. Metodas gali specifiikuoti mažiau išimčių, negu bazinė metodo versija arba visai jų negeneruoti (gali gaudyti išimtis)
2. BET NEGALI MESTI NAUJŲ Checked išimčių negu bazinė metodo versija

```
class A {  
    void m() throws CheckedException {  
    }  
}
```

```
class B extends A {  
    @Override  
    void m() throws CheckedException2{  
    }  
}
```



```
class CheckedException extends Exception {  
}
```

```
class CheckedException2 extends Exception {  
}
```

```
class SomeException extends RuntimeException {  
}
```

U6a Metodo perrašymas

Dvi klasės: antra paveldi pirmąją.

Pirma klasė metodas() throws MyException1{... - (checked exception)

Antra klasė perrašo šį metodą ir metodas() throws MyException2

--

```
class MyException1 extends Exception {}
```

```
class MyException2 extends Exception {}
```

Main pateistuoti

Apribojimai konstruktoriams

- Konstruktoriai gali generuoti bet kokias išimtis, t. y. apribojimai, kurie galioja
 - kitiems metodams, konstruktoriams netaikomi
- Paveldėtos klasės konstruktorius turi specifiikuoti ir visas bazinės klasės konstruktoriaus išimtis (nes paveldėtos klasės konstruktorius kviečia bazinės klasės konstruktorių), jei išimtys nepaveldi `RuntimeException`
- Paveldėtos klasės konstruktorius negali gaudyti bazinės klasės konstruktoriaus generuojamų išimčių

PVZ

```
class C {  
    C() throws SomeException { }  
    C(int i) throws CheckedException, SomeException { }  
}  
class D extends C {  
    D() throws CheckedException { }  
    D(int i) throws CheckedException {  
        super(i);  
    }  
}  
class CheckedException extends Exception {}  
class CheckedException2 extends Exception {}  
class SomeException extends RuntimeException {}
```

PVZ

```
class D extends C {  
  
    D() throws CheckedException {  
    }  
  
    D(int i) throws CheckedException {  
        try {  
            super(i);  
            // Constructor call must be the first statement in a constructor  
        } catch (Exception e) {  
  
        }  
    }  
}
```

Išimčių aptikimas

- Catch sakinyje galima nurodyti ne gaudomą, o bazinę išimtį
- Pirmiau nurodžius bazinę išimtį, o po to paveldėtą išimtį, kompiliatorius duos klaidą

Užd 7

- Sukurkime dvi klases (antra paveldi pirmą) jos turi po du konstruktorius.
- Pirmosios klasės
 - pirmas konstruktorius meta RuntimeException
 - antras - checked Exception
- Antros Klasės konstruktoriai kviečia pirmos klasės konstruktorius
- Konstruktoriuose išveskime tekstinius pranešimus

Main metode patestuoti, kaip viskas veikia

Užd 8

Sukurti metodą, kuris paprašytu vartotojo įvesti datą ir ją grąžintu.

Metodas grąžina Date objektą.

Main metode pateistuoti, kaip viskas veikia

Išimčių apdorojimas ND

MIF

-
1. Susikurkite tris išimtis E1, E2 ir E3, paveldėtas iš Exception. Susikurkite klasę A, turinčią metodus m1, m2, m3, m4 ir main. Iš metodo main kvieskite metodą m1, o iš metodo m1 – metodus m2, m3 ir m4. Metode m2 generuokite išimtį E1, metode m3 – E2, metode m4 – E3. Metode main įdėkite išimčių E1, E2 ir E3 gaudymą.
 2. Pakeiskite metodą main taip, kad būtų gaudomos bazinės klasės išimtys.
 3. Pakeiskite metodą main taip, kad išimtys būtų ne gaudomos, o išvedamos į konsolę.
 4. Klasėje A išimčių gaudytojus išdėstykite taip, kaip pavaizduota kitoje skaidrėje. Patikrinkite, ar ir dabar reikalingi visi sakiniai throws.
 5. Sukurkite klasę B, turinčią tuos pačius metodus, kaip ir klasė A. Išimčių generavimą pakeiskite realiais išimtis generuojančiais įvykiais: E1 keisti dalyba iš nulio, E2 – kreipimusi į masyvo elementą už masyvo ribų, E3 – kreipimusi į neegzistuojantį objektą (pvz., String tipo kintamajam priskirti null ir kviesti kokį nors String metodą). Atitinkamai modifikuokite išimčių gaudytojus. Patikrinkite, ar reikalingi sakiniai throws. Kodėl?

Išimčių apdorojimas (ND pav.)

