Accuracy and robustness

- **The basic philosophy of Java** is to **poorly written code will not execute!**

- It's a program **correct** if he successfully completes the task for which he was designed

- It's a program **robust** if it is able to react to unexpected situations (such as invalid input) in a reasonable way

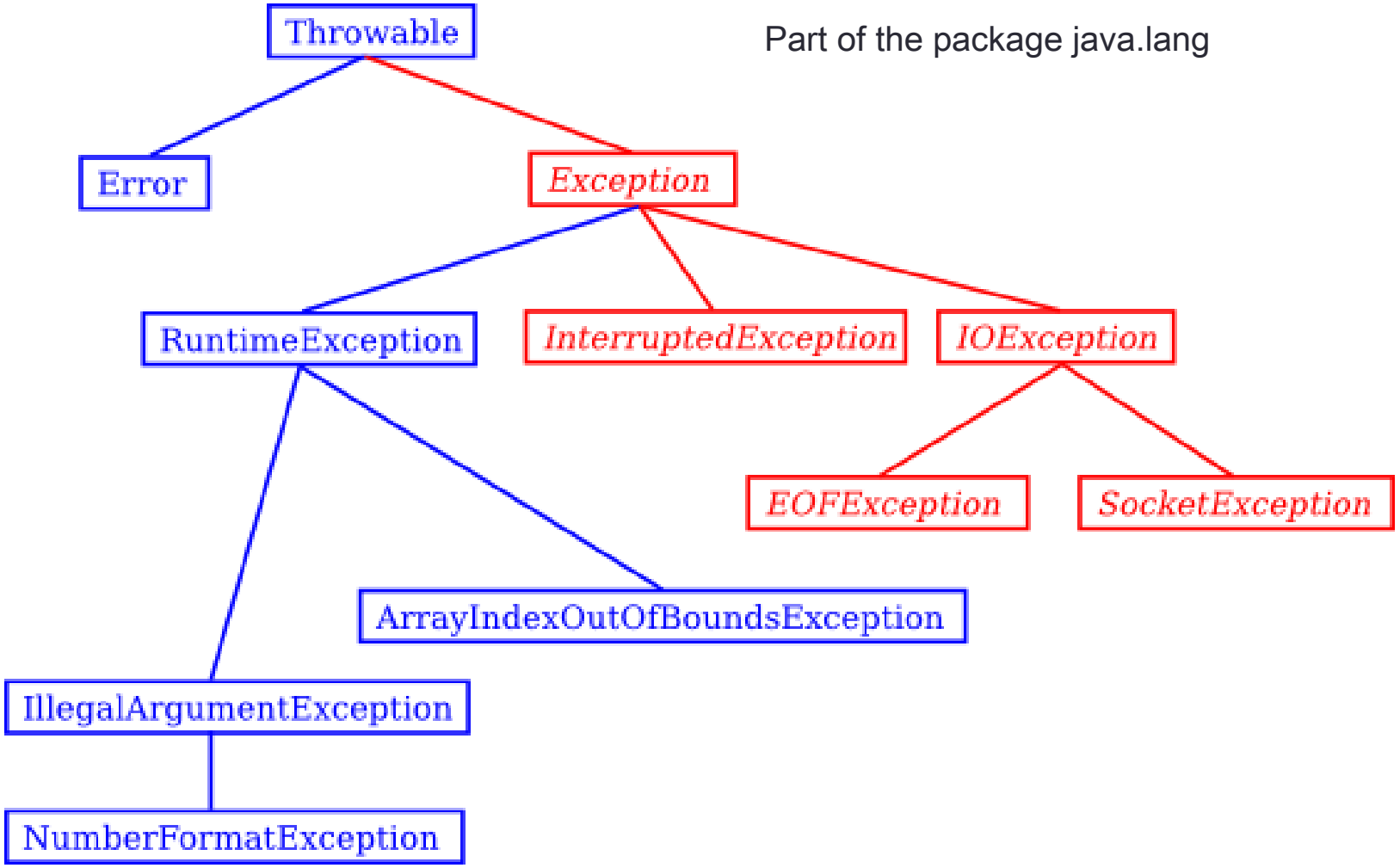
- To make **robust system, each of its components must be robust**

Dealing with mistakes

- Assuming that the code is able to detect and find the place where the error occurred, then it can be handled in several ways:

1. Ignoring a mistake is not a good way!
2. Check for possible problems and stop the program when a problem occurs
3. Checking for possible problems, catching a mistake and trying to solve the problem - trying to "recover"
4. Error handling using exceptions - "throwing" exceptions - desirable way!

Throwable and some of its subclasses



Error handling using exceptions

- Exceptions provide a way to detect errors and process them

- There is a special block for capturing and processing exceptions
- Block syntax for capturing and processing exceptions:

```
try {
        // here goes the potentially problematic code
} catch ( Exception e ) {
        // code that handles the exception
}
```

- If it's inside try block generated exception, block catch will process it.

Error handling using exceptions

- General form:

```
try {
                    // problematic code
} catch ( Exception e ) {
                    // code that handles the exception
                    System. out . println ( e .getMessage ());
}
System. out . println ( "Exception processed!" );
```

Example 4.1 - exception capture and processing

- The determinant of the matrix is equal to the difference of the product of the elements on the main and secondary diagonals

```
try {
        double determinant = M [0] [0] * M [1] [1] - M [0] [1] * M [1] [0];
        . . .
        System.out.println ( "Determinant M is" + determinant);
}
catch ( ArrayIndexOutOfBoundsException e) {
        System.out.println ( "M is not the correct size!" );
}
catch ( NullPointerException e) {
        System.out.print ( "Program error! M does not exist!" );
}
```

- If u try block generates an exception, then it is caught in the appropriate catch block

Error handling using exceptions

- If an exception occurs during block execution, the following happens:
  - Block execution is interrupted
  - The conditions from the clauses are checked catch (there may be more) to determine whether there is an appropriate one for that exception
    catch block
  - If none of catch blocks is not affected by that exception, then it is forwarded try higher level block - if an exception is not caught in the code it will be caught by the system with an unpredictable outcome!
  - If a suitable one is found catch block, commands from that block are executed
  - It then continues to execute the program starting from the first command behind the block try

General form of try-catch block

```
try {
   // code that can cause an exception
} catch (< tip1> <name1>) {
   // which handles type 1 exception
}
} catch (< tip2> <name1>) {
   // which handles type 2 exception
. . . [ finally { block}] // typically go here
                                    // activities that are always // performed,
                                    regardless of // whether an exception
                                    occurred
```

Cleaning with **finally**

- There are often pieces of code that we want to execute regardless of whether an exception occurred earlier or not
- This is usually the case in operations that do not involve memory recovery (eg we always want to close a previously opened file)
- So, in finally block go activities that always happen
- Since Java has a garbage collector, this command is necessary when we need to restore something other than memory to its original state - examples: closing a file, disconnecting from the network
- We will finally look at the block later in the example of working with files

Exception specification

- In Java, you are required to specify in the method specification which exceptions it can generate

- The word is reserved for that purpose **throws** **,,** e.g .:

```
void f () throws ArithmeticException, IOException {
        // code of a method that can cause exceptions
}
```

- If not specified in the specification **throws** **,,** it is assumed that the method does not generate exceptions
- Capturing any type of exception is done using the base class Exception - **catch** **(** Exception e) {…

Generate exception without processing

- There are situations when it makes sense to generate an exception without capturing and processing it
- When a program detects an error condition but there is no reasonable way to handle the error, then the program can generate an exception in the hope that some other part of the program will catch and process it. The keyword is reserved for this purpose **throw**
- Exceptions can be generated using conditions and **if** a block in which, if the condition is met, an exception is generated

Example 4.2 - generating an exception

```
/ **
  * Returns the greater than two roots of a quadratic equation
  * A * x * x + B * x + C = 0, if it has roots. If A == 0 or
  * is a discriminator B * B - 4 * A * C negative is then generated
  * IllegalArgumentException.
  * /
static public double root ( double A ,, double B ,, double C )
                                    throws IllegalArgumentException {
            if ( A == 0) {
                        throw new IllegalArgumentException ( "And it can't be zero!" );
            }
            else {
                        double disk = B * B - 4 * A * C ;
                        if ( disk <0)
                                    throw new IllegalArgumentException ( "Discriminant
                                                                                less than zero! " );
                        return (- B + Math. sqrt ( disk )) / (2 * A );
            }
}
```

Instructions for using exceptions

- We use exceptions to:
  - Let's solve the problems and call again the method that caused the exception
  - Let's "fix" the error and continue working without trying the method again
  - Let's simplify the code (if the way we generate exceptions complicates the code even more, then it is not easy and desirable to use it)
  - Let's finish the program
  - We increase the reliability of the library and programs - short-term effort in writing debugging code is a long-term investment in the strength and stability of the application

Assertions

- Assertions ensure that a prerequisite is met to allow further execution of the program

- A reserved word is used **assert**
- Command forms:

  assert condition;

  assert condition: error_message;

- Enabling assumptions in Eclipse: - Run As - Run Configurations ... - Arguments tab - VM arguments - “-ea”
- Example:

```
assert ( fact == 1): "Factorial not initialized to 1!" ;
```

Example 4.3 - Assumptions

```
/ **
  * Returns the greater than two roots of a quadratic equation
  * A * x * x + B * x + C = 0, if it has roots.
  * Prerequisites: A! = 0 and B * B - 4 * A * C> 0
  * /
static public double root ( double A, double B, double C) {
                assert A! = 0: “Leading coefficient of the quadratic equation
                                                we must not be zero! " ;
                double disk = B * B - 4 * A * C;
                assert disk> = 0: "The discriminant of the quadratic equation
                                                    we must not be negative! " ;
                return (- B + Math.sqrt (disk)) / (2 * A);
            }
```

Annotations

- Annotations are metadata

- Java 5 stations - notes are checked by the compiler to ensure that the code is in accordance with the programmer's intentions
- Examples:
  - @Override
  - @Deprecated
  - @SuppressWarnings
- They are used in the code very similarly to static, final etc. - if it is written e.g. @ Override in the definition of a method, then it should redefine the method of the same name from a superclass - if such a method does not exist the compiler reports an error!

WORKING SATOKOVIM I

FILES