Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Programming 1 with Java

User Input Validation and Exceptions

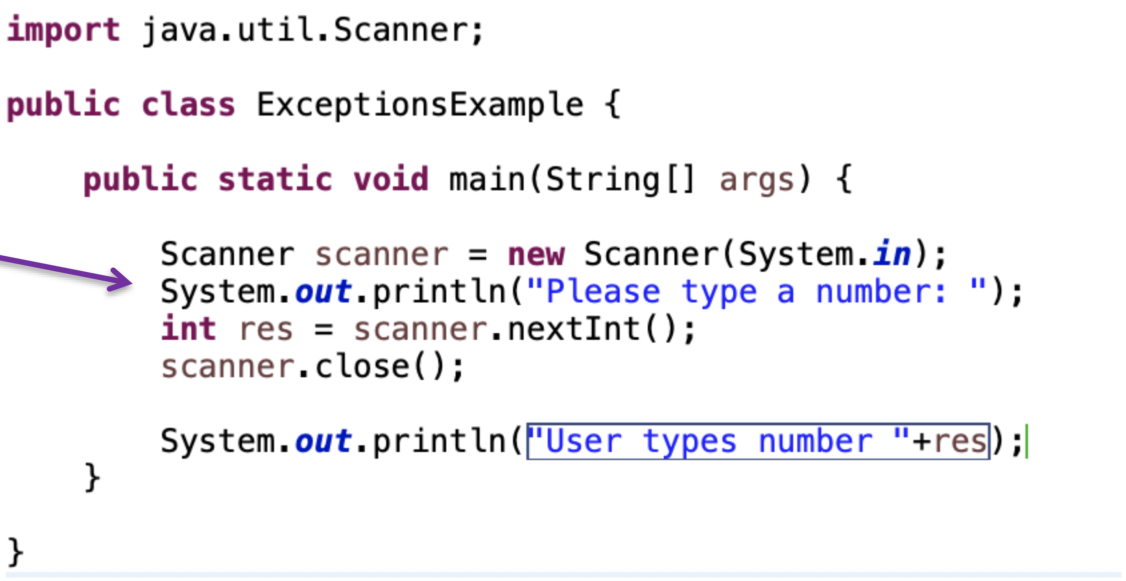
Prof. Dr. Mascha Kurpicz-Briki

HS 2019

When Things Go Wrong…

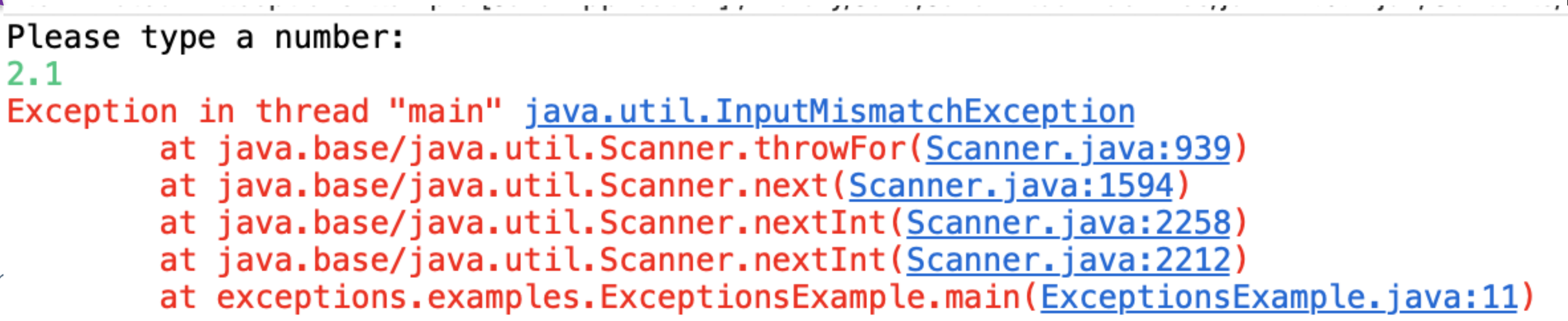
Sometimes, the user does not behave as we expect…

Expectation: Integer number (1 or 10 or 350…)



```
import java.util.Scanner;

public class ExceptionsExample {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Please type a number: ");
        int res = scanner.nextInt();
        scanner.close();

        System.out.println("User types number "+res);
    }

}
```

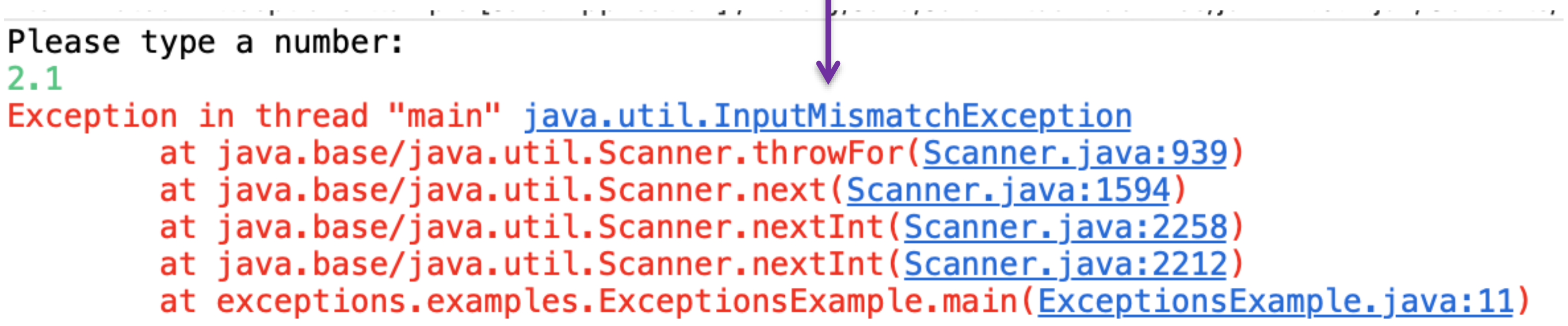
User misunderstood and types 2.1



```
Please type a number:
2.1
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at exceptions.examples.ExceptionsExample.main(ExceptionsExample.java:11)
```

Exception

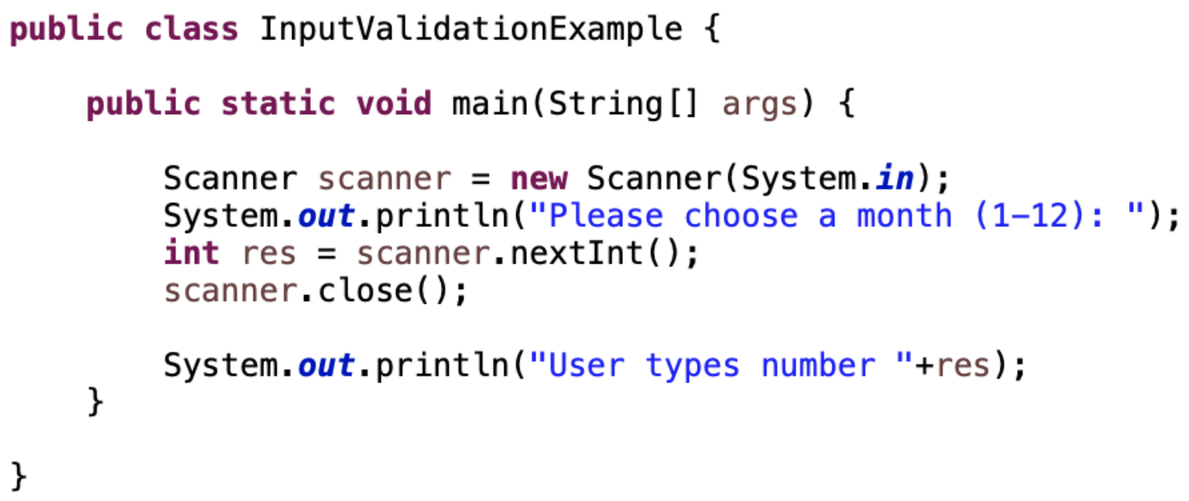
Exception Type



```
Please type a number:
2.1
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at exceptions.examples.ExceptionsExample.main(ExceptionsExample.java:11)
```

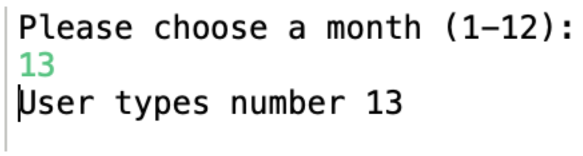
Program crashes with StackTrace: Hard to read, not very useful for the user!

What else can go wrong?



```
public class InputValidationExample {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Please choose a month (1-12): ");
        int res = scanner.nextInt();
        scanner.close();

        System.out.println("User types number "+res);
    }

}
```

No exception, but does it make any sense?



```
Please choose a month (1-12):
13
User types number 13
```

This can lead to problems later in the program execution!

In the Worst Case…

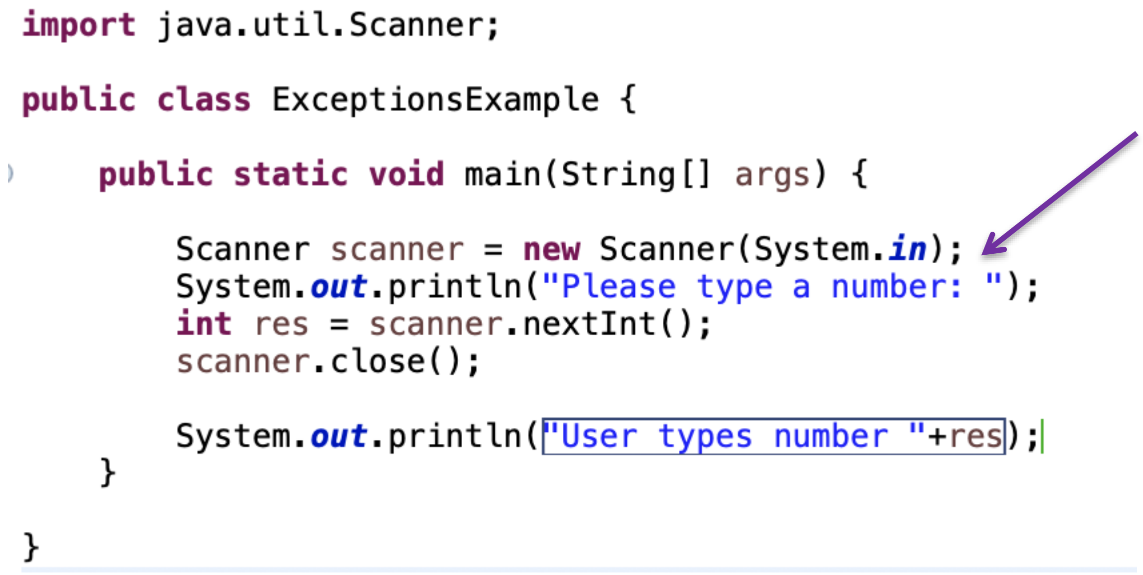
Explosion of the Ariane 5 1996

http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html

Exception Handling

What are Exceptions?

An exception is an event, which occurs during the execution of a program, that **disrupts the normal flow** of the program.



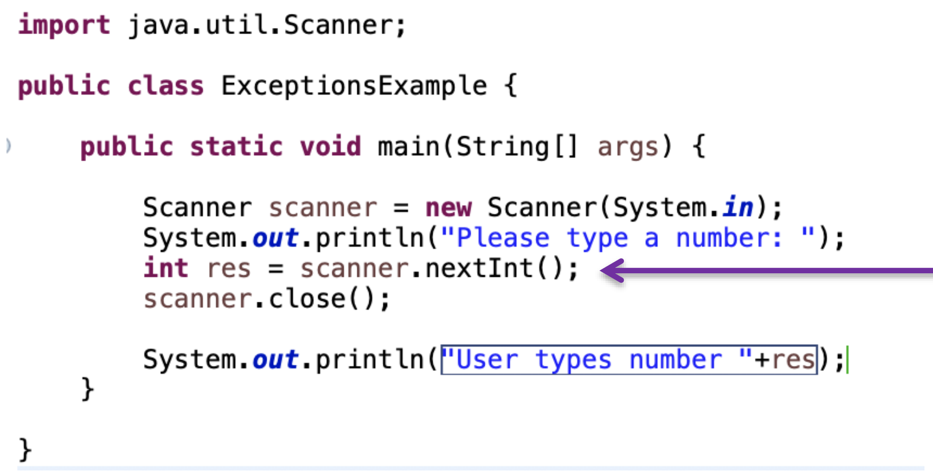
```java
import java.util.Scanner;

public class ExceptionsExample {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Please type a number: ");
        int res = scanner.nextInt();
        scanner.close();

        System.out.println("User types number "+res);
    }

}
```

„Normal flow“: User types an integer number and all goes well…

What happens when an Error occurs?

- When an error occurs within a method, the method creates an object and hands it off to the runtime system
- This exception object contains information about the error, including its type and the state of the program when the error occurred
- Creating an exception object and handing it to the runtime system is called throwing an exception

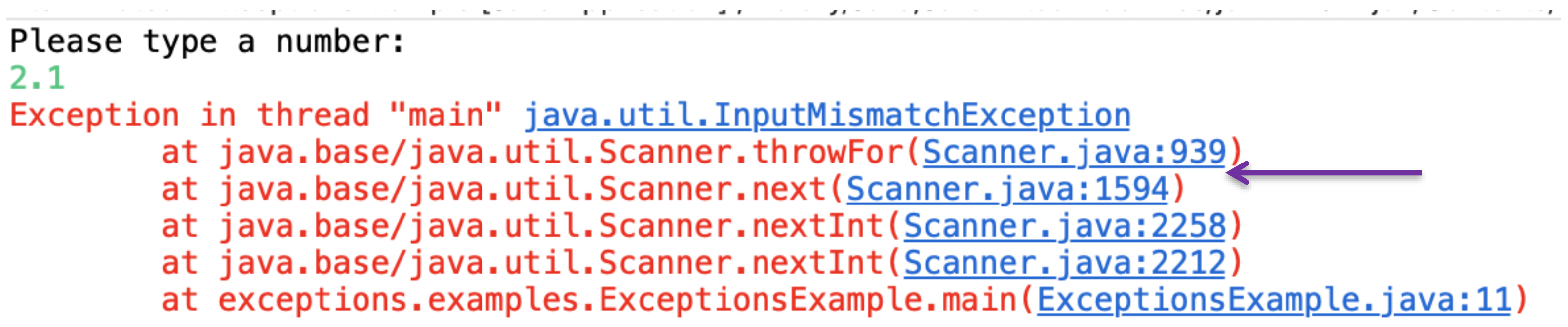
What happens when an Error occurs?



```
import java.util.Scanner;

public class ExceptionsExample {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Please type a number: ");
        int res = scanner.nextInt();
        scanner.close();

        System.out.println("User types number "+res);
    }

}
```

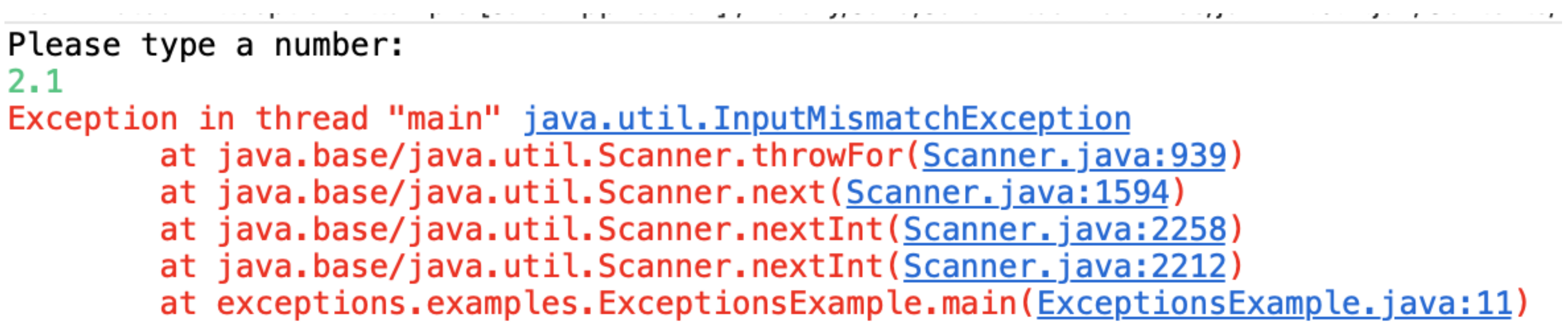
1) Problem is detected, **exception object** is created and passed to the runtime system

2) Program is stopped, because of the exception



```
Please type a number:
2.1
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at exceptions.examples.ExceptionsExample.main(ExceptionsExample.java:11)
```

Exception Handling



```
Please type a number:
2.1
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at exceptions.examples.ExceptionsExample.main(ExceptionsExample.java:11)
```

Exception was thrown in the Scanner class, and passed until the main method of our ExceptionExample class, because nobody **handled** it

Error Handling in Java

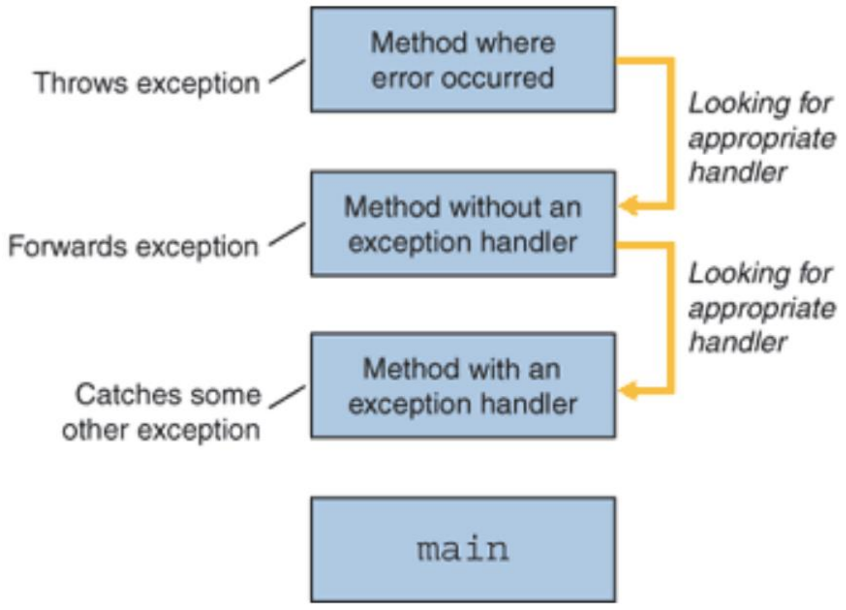

Method where error occurred
Throws exception
Looking for appropriate handler
Method without an exception handler
Forwards exception
Looking for appropriate handler
Method with an exception handler
Catches some other exception
main

Fig.: Call Stack (upside down)
Source: java.sun.com

Scanner throws an exception: the normal flow of the program is disrupted because of invalid input

It cannot do anything to remedy the situation, so it passes the exception upward the hierarchy in the call stack.

Error Handling in Java

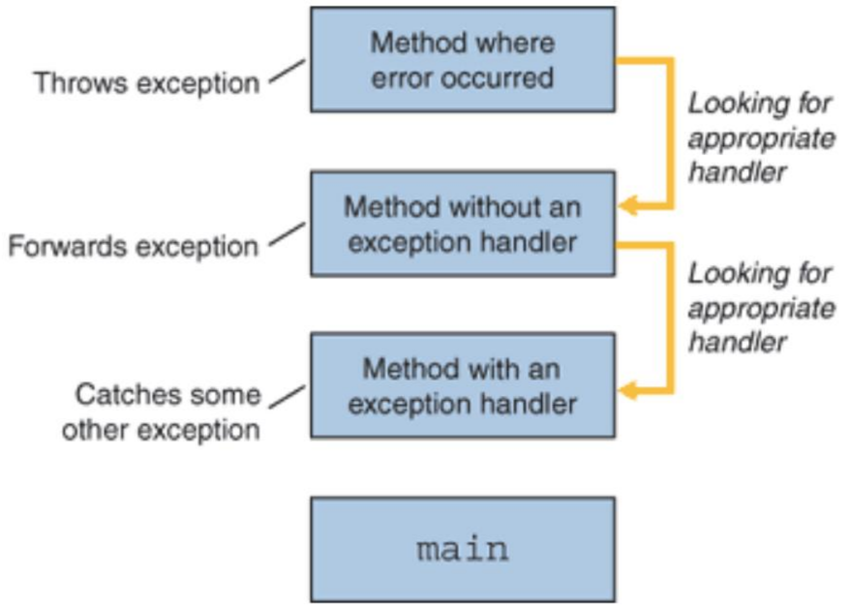
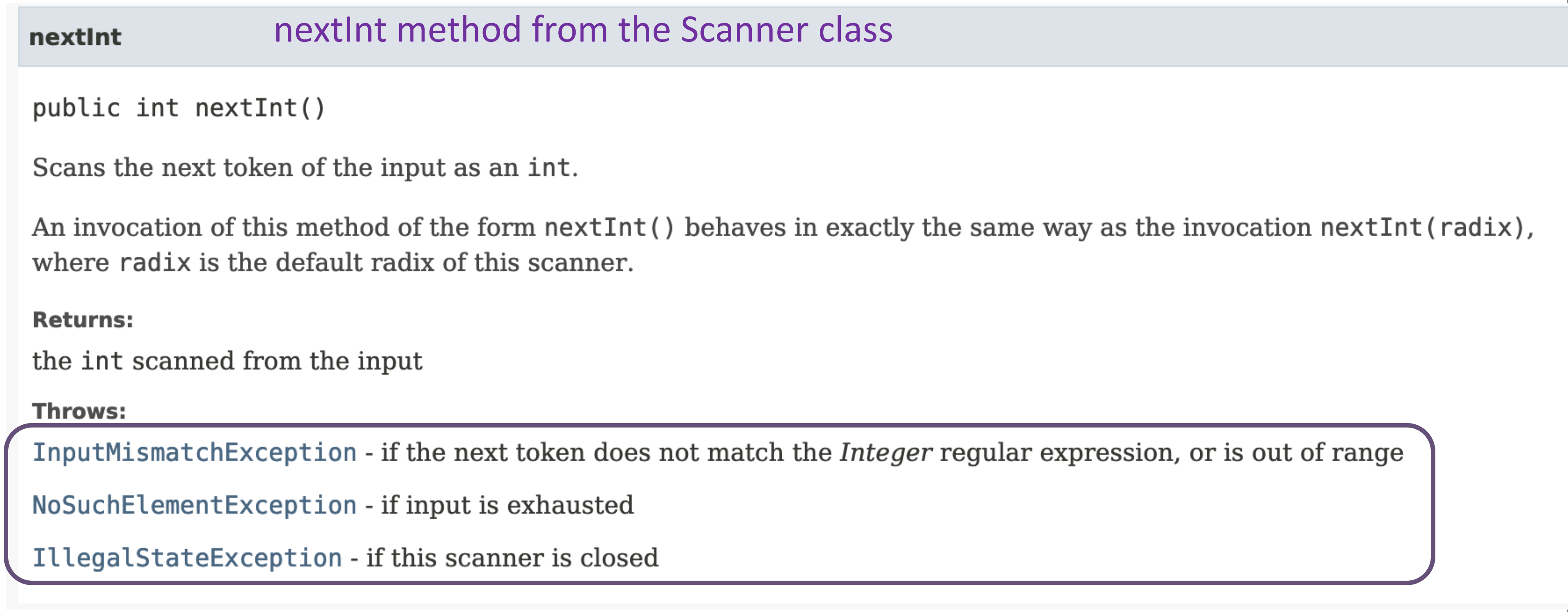

Method where error occurred
Throws exception
Looking for appropriate handler
Method without an exception handler
Forwards exception
Looking for appropriate handler
Method with an exception handler
Catches some other exception
main

Fig.: Call Stack (upside down)
Source: java.sun.com

Finally, the exception comes to the main method in the ExceptionsExample class

Here we should handle it, because we can interact with the user, and tell him to put a correct value.

Handling an Exception



nextInt nextInt method from the Scanner class

```
public int nextInt()
```

Scans the next token of the input as an `int`.

An invocation of this method of the form `nextInt()` behaves in exactly the same way as the invocation `nextInt(radix)`, where `radix` is the default radix of this scanner.

Returns:

the `int` scanned from the input

Throws:

`InputMismatchException` - if the next token does not match the *Integer* regular expression, or is out of range

`NoSuchElementException` - if input is exhausted

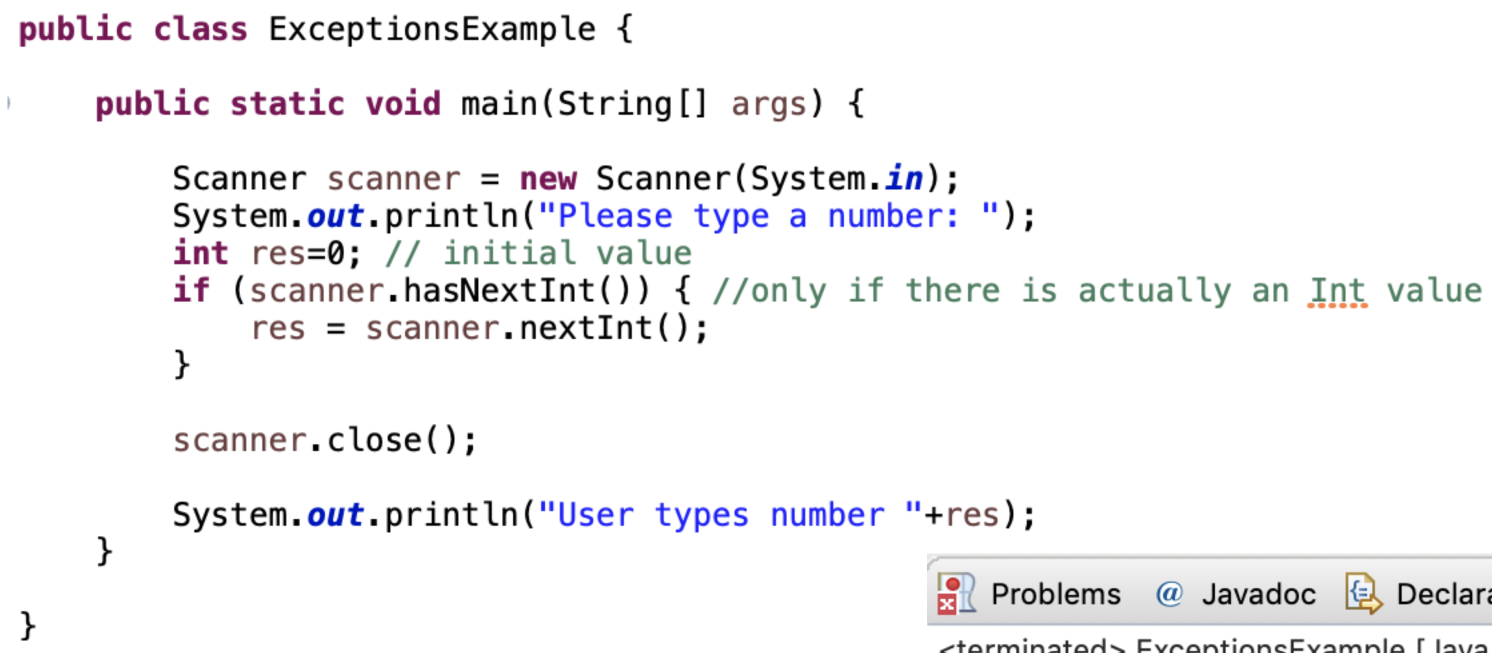
`IllegalStateException` - if this scanner is closed

Javadoc tells us, what type of exceptions can be thrown by a specific method.

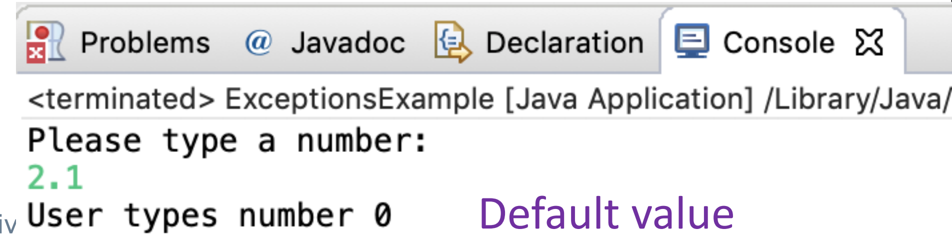
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Scanner.html#nextInt()

Careful Programming

- Sometimes, we can avoid exceptions by programming carefully
- Typically, this is better than handling exceptions



```java
public class ExceptionsExample {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Please type a number: ");
        int res=0; // initial value
        if (scanner.hasNextInt()) { //only if there is actually an Int value
            res = scanner.nextInt();
        }

        scanner.close();

        System.out.println("User types number "+res);
    }

}
```



```
<terminated> ExceptionsExample [Java Application] /Library/Java/
Please type a number:
2.1
User types number 0
```

Default value

When we cannot avoid it, handle it (Option 1)

Handle it locally with try/catch block

```
Scanner scanner = new Scanner(System.in);
System.out.println("Please type a number: ");
int res = 0; // initial value
try {
res = scanner.nextInt();
} catch (InputMismatchException ex) {
    // Do something useful if this exception occurs!
}
```

try: This code might throw an exception

catch: This is how we handle the exception

When we cannot avoid it, handle it (Option 2)

Pass it on in the call stack hierarchy and handle it elsewhere…

readInput method **throws** an exception

```
public class ExceptionsExampleHandled2 {

    public static int readInput(Scanner scanner) throws InputMismatchException {
        return scanner.nextInt();
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Please type a number: ");
        int res = 0; // initial value
        try {
        res = readInput(scanner);
        } catch (InputMismatchException ex) {
            // Do something useful if this exception occurs!
        }

        scanner.close();

        System.out.println("User types number " + res);
    }

}
```

main method **handles** the exception

How to Choose the Best Option?

- Handle exceptions as local as possible
- Throw exceptions further up, if you need more information about the context to handle them

Hints

A method can throw multiple types of exception

```
public void read(String filename) throws IOException,
ClassNotFoundException
```

Hints

We can provide several catch-blocks for different types of exceptions that can happen

```
Scanner scanner = new Scanner(System.in);
System.out.println("Please type a number: ");
int res = 0; // initial value
try {
res = scanner.nextInt();
} catch (InputMismatchException ex) {
    // Do something useful if InputMisMatchException occurs!
} catch (NoSuchElementException ex) {
    // Do something useful if NoSuchELementException occurs!
}
```

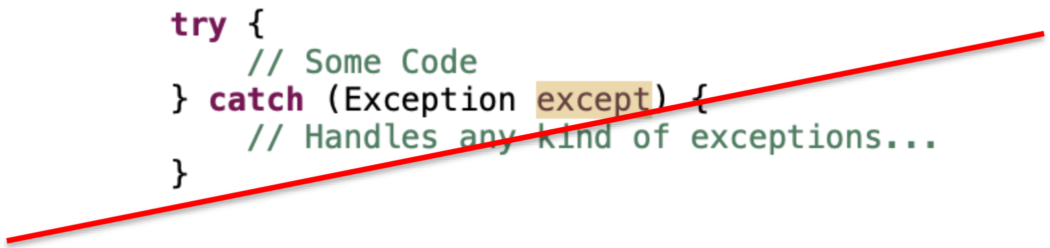
Hints

A catch-block can be used for different types of exceptions

```
Scanner scanner = new Scanner(System.in);
System.out.println("Please type a number: ");
int res = 0; // initial value
try {
res = scanner.nextInt();
} catch (InputMismatchException ex) {
    // Do something useful if InputMisMatchException occurs!
} catch (NoSuchElementException|IllegalStateException ex) {
    // Do something useful if NoSuchELementException or IllegalStateException occurs!
}
```

Hints

Be specific when handling exceptions

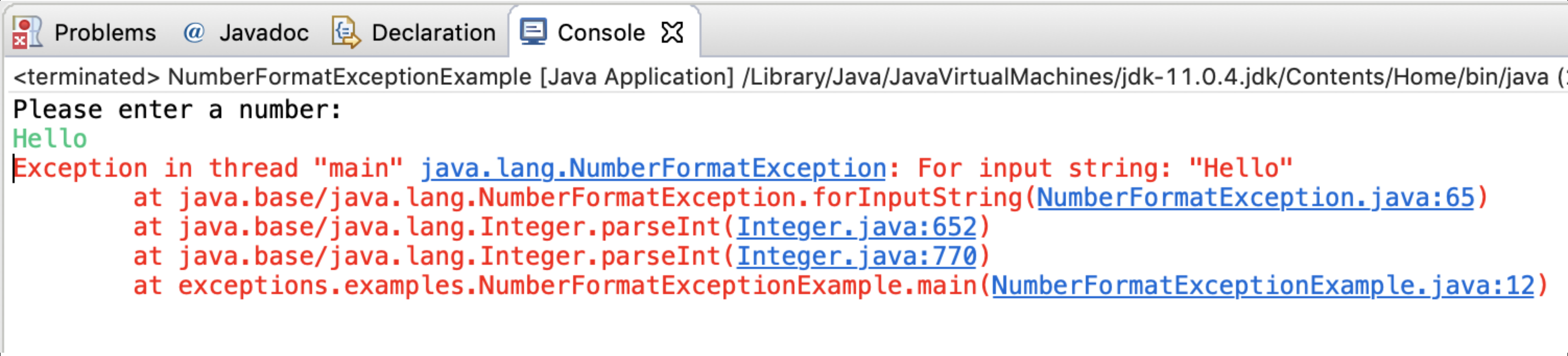


```
try {
    // Some Code
} catch (Exception except) {
    // Handles any kind of exceptions...
}
```

Example: NumberFormatException

```
public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);
    System.out.println("Please enter a number: ");
    String input = scanner.nextLine();
    int number = Integer.parseInt(input); // Read an int from the input (which is a String)
    scanner.close();

    System.out.println("User types number "+number);
}
```

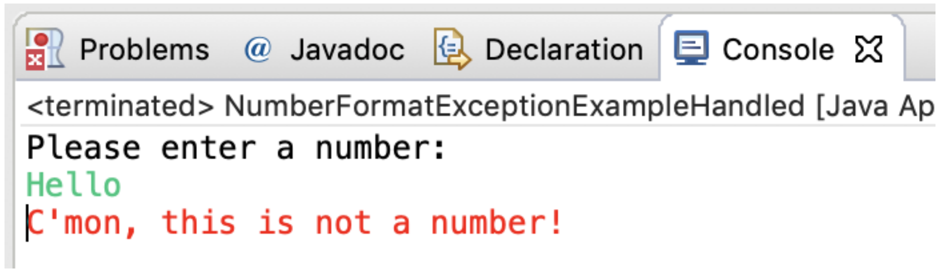
Never trust the user…be ready for any kind of input…



```
<terminated> NumberFormatExceptionExample [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java (
Please enter a number:
Hello
Exception in thread "main" java.lang.NumberFormatException: For input string: "Hello"
	at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
	at java.base/java.lang.Integer.parseInt(Integer.java:652)
	at java.base/java.lang.Integer.parseInt(Integer.java:770)
	at exceptions.examples.NumberFormatExceptionExample.main(NumberFormatExceptionExample.java:12)
```

Example: NumberFormatException

```
Scanner scanner = new Scanner(System.in);
System.out.println("Please enter a number: ");
try {
String input = scanner.nextLine();
int number = Integer.parseInt(input); // Read an int from the input (which is a String)
} catch (NumberFormatException ex) {
    System.err.println("C'mon, this is not a number!");
}
```



```
<terminated> NumberFormatExceptionExampleHandled [Java Ap
Please enter a number:
Hello
C'mon, this is not a number!
```

Checked vs. Unchecked Exceptions

- Not every exception a method may throw must be declared. → Only checked exceptions.
- Checked Exception
  - Extend the class `Exception`.
  - The compiler checks that you do not ignore them.
  - Usually, the application should be able to recover from them.
  - Many checked exception occur when dealing with input/output (user input, file handling, communication)
  - Examples: `IOException`, `NumberFormatException`, ...

- Unchecked Exceptions
  - Extend the class `RuntimeException` or `Error`.
  - They are the programmer's fault. Application are not expected to recover from these errors.
  - Examples: `ArithmethicException`, `NullPointerException`, `IllegalArgumentException`, `IllegalStateException`, `IndexOutOfBoundsException`, ...

Find Exception Types

- Exceptions are objects and therefore you can find possible types of exceptions in the Java API Documentation

- For example, you find details about the IOException:

https://docs.oracle.com/javase/10/docs/api/java/io/IOException.html

Good Idea: Finally Clause

```
try {
res = scanner.nextInt();
} catch (InputMismatchException ex) {
    // Do something useful if InputMisMatchException occurs!
} catch (NoSuchElementException|IllegalStateException ex) {
    // Do something useful if NoSuchELementException or IllegalStateException occurs!
}

finally {
    scanner.close();
}
```

If an exception occurs or not, the finally block is **always** executed. We want the scanner to close the scanner in any case.

Validating User Input

Input Validation

- So far we have been very sloppy with handling user input.
- We have not validated whether the input is correct.
- We need to check the valid range of a value.

don't add new line at the end

```
Scanner scanner = new Scanner(System.in);
System.out.print("In which month were you born (2: February, ...)? ");
int month = scanner.nextInt();
...
LocalDate birthday = LocalDate.of(year, month, day);
```

Assume user enters incorrect month number "14".
LocateDate verifies correctness of arguments and throws Exception → program crashes.

```
In which month were you born (2: February, ...)? 14
Exception in thread "main" java.time.DateTimeException: Invalid value for MonthOfYear
(valid values 1 - 12): 14
at java.time.temporal.ValueRange.checkValidValue(ValueRange.java:311)
at java.time.temporal.ChronoField.checkValidValue(ChronoField.java:703)
at java.time.LocalDate.of(LocalDate.java:267)
at ValidateInput.main(ValidateInput.java:13)
```

Stack Trace shows call stack

Input Validation

- We need to check the value entered is of the correct type.
- If we expect an int, the user must not enter a string that cannot be converted to an int.

```
Scanner scanner = new Scanner(System.in);
System.out.print("In which month were you born (2: February, ...)? ");
int month = scanner.nextInt();
...
LocalDate birthday = LocalDate.of(year, month, day);
```

Assume user enters incorrect month number “March”.
nextInt() of Scanner verifies that next token can be parsed as an int, otherwise, throws an exception.

```
In which month were you born (2: February, ...)? March
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:864)
at java.util.Scanner.next(Scanner.java:1485)
at java.util.Scanner.nextInt(Scanner.java:2117)
at java.util.Scanner.nextInt(Scanner.java:2076)
at ValidateInput.main(ValidateInput.java:9)
```

Exceptions while parsing number from String

- Often, strings need to parsed into numbers.
- Use
  - `Byte.parseByte(aString)` to parse a `byte`
  - `Short.parseShort(aString)` to parse a `short`
  - `Integer.parseInt(aString)` to parse an `int`
  - `Long.parseLong(aString)` to parse a `long`
  - `Float.parseFloat(aString)` to parse a `float`
  - `Double.parseDouble(aString)` to parse a `double`

- If number cannot be parsed, the methods throw a `NumberFormatException`.
- Incorrect user input is not an unexpected error, so your program should to handle this error. Either recover from the error or return suitable error message back to the user. An exception with stack trace is not sufficient. → catch `NumberFormatExceptions`.

Example: Validating User Input and Handling Errors

```
System.out.print("In which month were you born (2: February, ...)? ");
String monthStr = scanner.next();  // get next token as a string
int month = 0;
try {  // try to parse int
  month = Integer.parseInt(monthStr);
} catch (NumberFormatException e) {
  System.err.println("Enter a valid number. Got " + monthStr);
  System.exit(1); // terminate program with an non-zero error code
}
if (month < 1 || month > 12) {  // check that int is in range
  System.err.println("Invalid month entered " + month + ", valid 1 .. 12.");
  System.exit(1);
 }
```

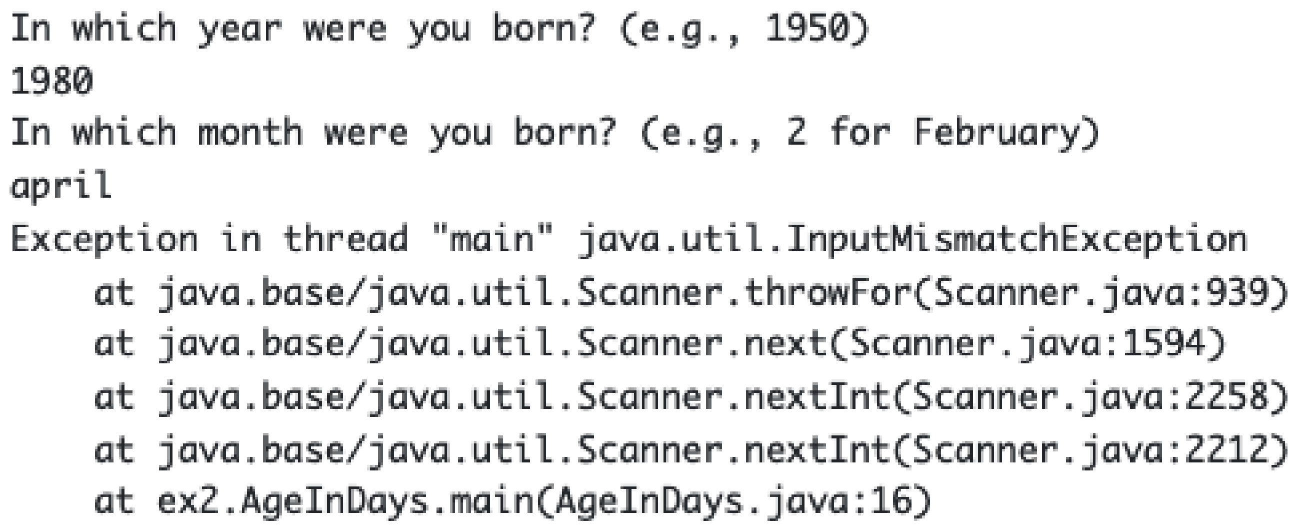
`System.err` is error output (instead of standard output)
Error output is shown red in Eclipse console.

Example: Catching the Scanner Exception

```
System.out.print("In which month were you born (2: February, ...)? ");
int month = 0;
try {  // try extract an int from the scanner
  month = scanner.nextInt();
} catch (InputMismatchException e) {
  System.err.println("Enter a valid number.");
  System.exit(1); // terminate program with an non-zero error code
}
if (month < 1 || month > 12) {  // check that int is in range
  System.err.println("Invalid month entered " + month +
      ", valid 1 .. 12.");
  System.exit(1);
 }
```

Exercise 1: Age in Days

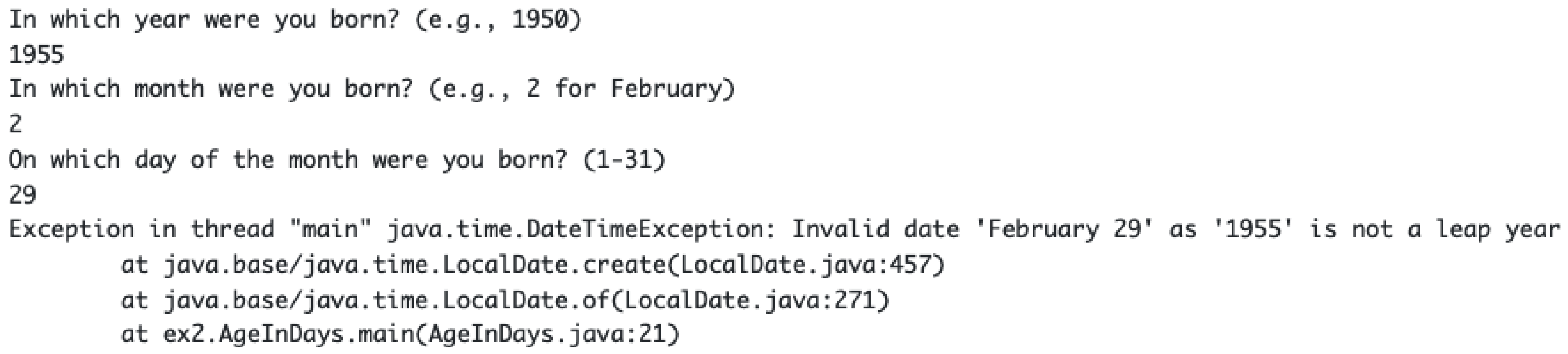
- Download AgeInDays.java. The program calculates the age of a person from the date of birth that the user enters from the console. An incorrect input results in an exception in the scanner:



```
In which year were you born? (e.g., 1950)
1980
In which month were you born? (e.g., 2 for February)
april
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at ex2.AgeInDays.main(AgeInDays.java:16)
```

Exercise 1: Age in Days

- If the user enters an invalid date, for example, February 29 and a non-leap year, an exception is also thrown in LocalDate. Consider this input for example:



```
In which year were you born? (e.g., 1950)
1955
In which month were you born? (e.g., 2 for February)
2
On which day of the month were you born? (1-31)
29
Exception in thread "main" java.time.DateTimeException: Invalid date 'February 29' as '1955' is not a leap year
	at java.base/java.time.LocalDate.create(LocalDate.java:457)
	at java.base/java.time.LocalDate.of(LocalDate.java:271)
	at ex2.AgeInDays.main(AgeInDays.java:21)
```

Exercise 1: Age in Days

- Tasks:
  - In this problem, extend your solution such that the numbers entered by the user are validated and appropriate error messages returned when:
    - the user input cannot be parsed into an integer
    - the number entered is out of range, e.g., a month less than 1 or greater than 1, or a day less than 1 or greater than 31
    - or a day-month-year combination that results in an invalid date, like February 29, 1955, is entered.
- Your code should return an appropriate error message when the user enters invalid data. After the error is reported, terminate the program with an error code by invoking System.exit(1) , where any non-zero value (1) stands for a error return code.

Exercise 1: Age in Days

- Hints:
  - Your code needs to check inputs for three inputs, day, month, and year. Avoid duplication of your code by extracting the logic that checks the user input into a separate method. Invoke this method once for each number in the birth day date.
  - When something goes wrong when parsing an integer from a String, you can use NumberFormatException to catch it.
  - When something goes wrong with LocalDate.of(year,month,day) because the entered date is invalid, you can use DateTimeException to catch it.

Exercise 2: Extend BankAccountCLI (optional)

- Implement user input validation and exception handling for the BankAccountCLI
- Think of potential mistakes, that the user can make and avoid them