Technische Hochschule Rosenheim
Technical University of Applied Sciences

Programming Basics – WiSe21/20

Exceptions

Prof. Dr Silke Lechner-Greite

Table of contents – planned topics

Technische Hochschule Rosenheim

Technische Hochschule Rosenheim

Chapter 11: Characters and Strings

Initial situation

Technische Hochschule Rosenheim

- Occurrence of errors during programme development:
  - Compilation errors
    - Problem with the syntax of the programme
    - During compilation, the compiler finds them → non-critical
  - Runtime errors
    - Problems that occur during programme execution
    - Possible causes: logical errors in the programme, incorrect use (operating error) of a correct programme, file or network operations, problems in the Java runtime system, …
- Wanted:
  - Linguistic mechanism for a controlled reaction to runtime errors
  - Exceptions

Idea

Technische Hochschule Rosenheim

- Implement appropriate response, depending on the type of error
  - Logical error => stop programme
  - Operating error => notification, prompt for correction
  - Problem in JVM => hardly any useful measures
  - …
- The use of exceptions has two main advantages:
  - Runtime errors (triggered exceptions) cannot be ignored, measures must be taken for handling (otherwise: termination of the programme)
  - Code for the regular programme execution is textually separated from the code for error handling

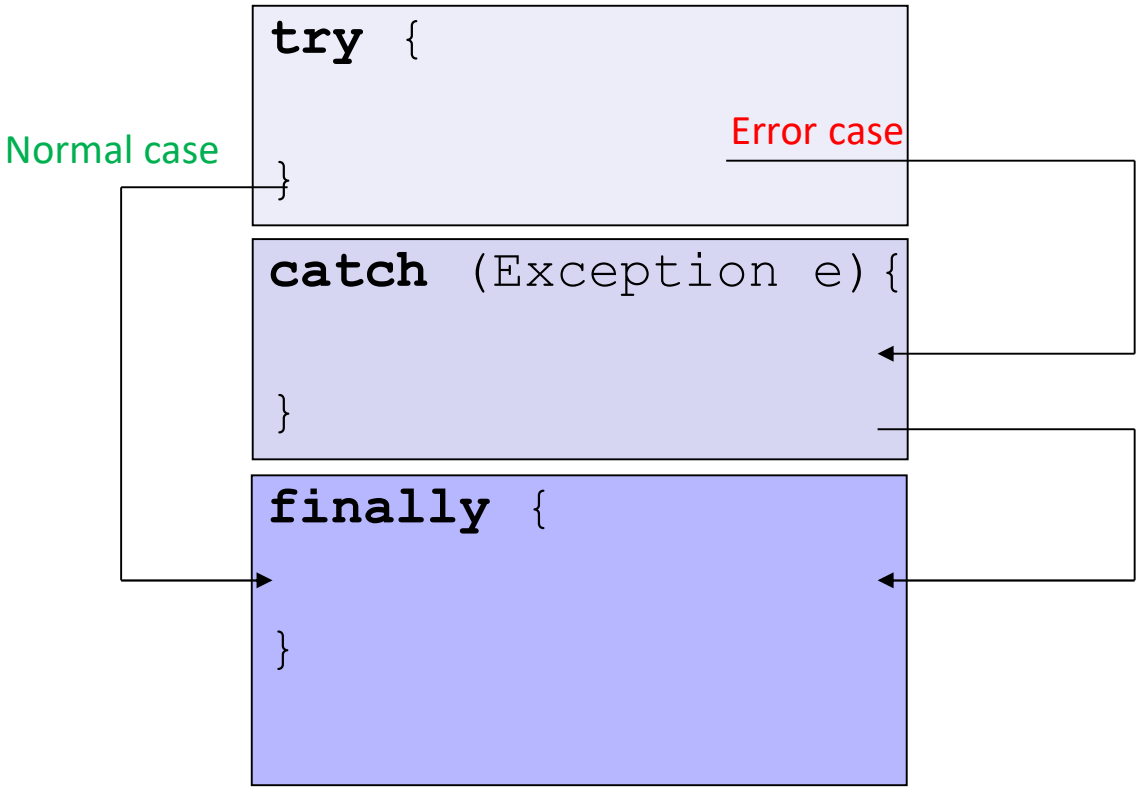
Method

Technische Hochschule Rosenheim

- Possibilities for programmers:
  - By including the critical statements in a `try` block, possible error situations can be checked at runtime.
  - In case of an error, branch to a `catch` block.
  - If completion work is required in both normal cases and cases of error, then a so-called `finally` block can be added.

Control flow for `try-catch-finally` statement

Technische Hochschule Rosenheim



Example `try-catch-finally` statement

Technische Hochschule Rosenheim

```
public void checkInput(String s) {
    try {
       // change the string into an integer
       int myInt = Integer.parseInt(s);
    }
    catch(NumberFormatException e) {
       // if the conversion went wrong...
       // give the user a notification
       System.out.println("Not a valid number:" + s);
    }
    finally { // is always executed after try (and catch)
       // in every case...
       // write the entry to console
       System.out.println("Entry was: " + s);
    }
}
```

Local exception handling

Exception handling in calling method (1)

Technische Hochschule Rosenheim

- In many cases:
  - Exception handling should not be done in the method in which the exception occurred, but in the calling method
- Required:
  - Calling method must be informed of the exception
  - Possibility to pass an exception that has occurred to the calling method
- Linguistic mechanism in Java:
  - Add a `throws` clause to the method declaration
  - Method body will trigger an exception

Exception handling in calling method (2)

Technische Hochschule Rosenheim

- Example:
  - Method `clip()` gets a string
  - Truncates first and last character, returns the rest

```
String clip(String s) {
  return s.substring(1, s.length() - 1);
}
```

- Possible exception situations:
  - No string is passed (**`null`** reference)
  - String is too short (less than two characters)

```
String clip(String s) {
  if (s == null)                    // no string?
    ...
  else if (s.length() < 2)          // string too short?
    ...
  else
    return s.substring(1, s.length() - 1);
}
```

Exception handling in calling method (3)

Technische Hochschule Rosenheim

- Add a `throws` clause to the method declaration
- In exception situations, an exception is triggered ("throw an exception" that is "caught" elsewhere)
  - Statement to throw an exception: **`throw`** `<exception>;`
  - Type of expression `<exception>` must be compatible with the predefined class `Throwable`
  - Initially sufficient: Object of the class `Exception`, which is derived from `Throwable`

```
String clip(String s) throws Exception {
  if (s == null)
    throw new Exception();            // throw an exception
  else if (s.length() < 2)
    throw new Exception();            // throw an exception
  else
    return s.substring(1, s.length() - 1);
}
```

Exception handling in calling method (4)

Technische Hochschule Rosenheim

- Procedure
  - `throw` interrupts the execution of the source code immediately
  - The subsequent statements are no longer executed (`throw` acts similar to a `return`)
  - `else` thus unnecessary in the example
- Implementation variant 2:

```
String clip(String s) throws Exception {

  if (s == null)
    throw new Exception();           // throw an exception

  if (s.length() < 2)
    throw new Exception();           // throw an exception

  return s.substring(1, s.length() - 1);
}
```

Exception handling in calling method (5)

Technische Hochschule Rosenheim

- Create an exception object
  - Constructor of the `Exception` class accepts string parameters
  - Additional information about the cause of the error
  - Intended for users of the method or programme
  - Not for automatic evaluation
- Implementation variant 3:

```
String clip(String s) throws Exception {

  if (s == null)
    throw new Exception("no string");  // throw an exception

  if (s.length() < 2)
    throw new Exception("string too short"); // throw an exception

  return s.substring(1, s.length() - 1);
}
```

Exercise – Trigger exceptions

Technische Hochschule Rosenheim

- Live exercise
  - Complete Task 1 on the live exercises sheet “Exceptions”
  - You have 5 minutes.

Exception handling in calling method (6)

Technische Hochschule Rosenheim

- Call the `clip` method in another method

```
public static void main (String[] args) {
  String s = "Test string";

  try {
    System.out.println(clip(s));
  }
  catch (Exception ex) {
       System.out.println("clip method failed: "
       + ex.getMessage());
}
```

Exercise – Handling exceptions

Technische Hochschule Rosenheim

- Live exercise
  - Complete Task 2 on the live exercises sheet “Exceptions”
  - You have 5 minutes per part.

Technische Hochschule Rosenheim

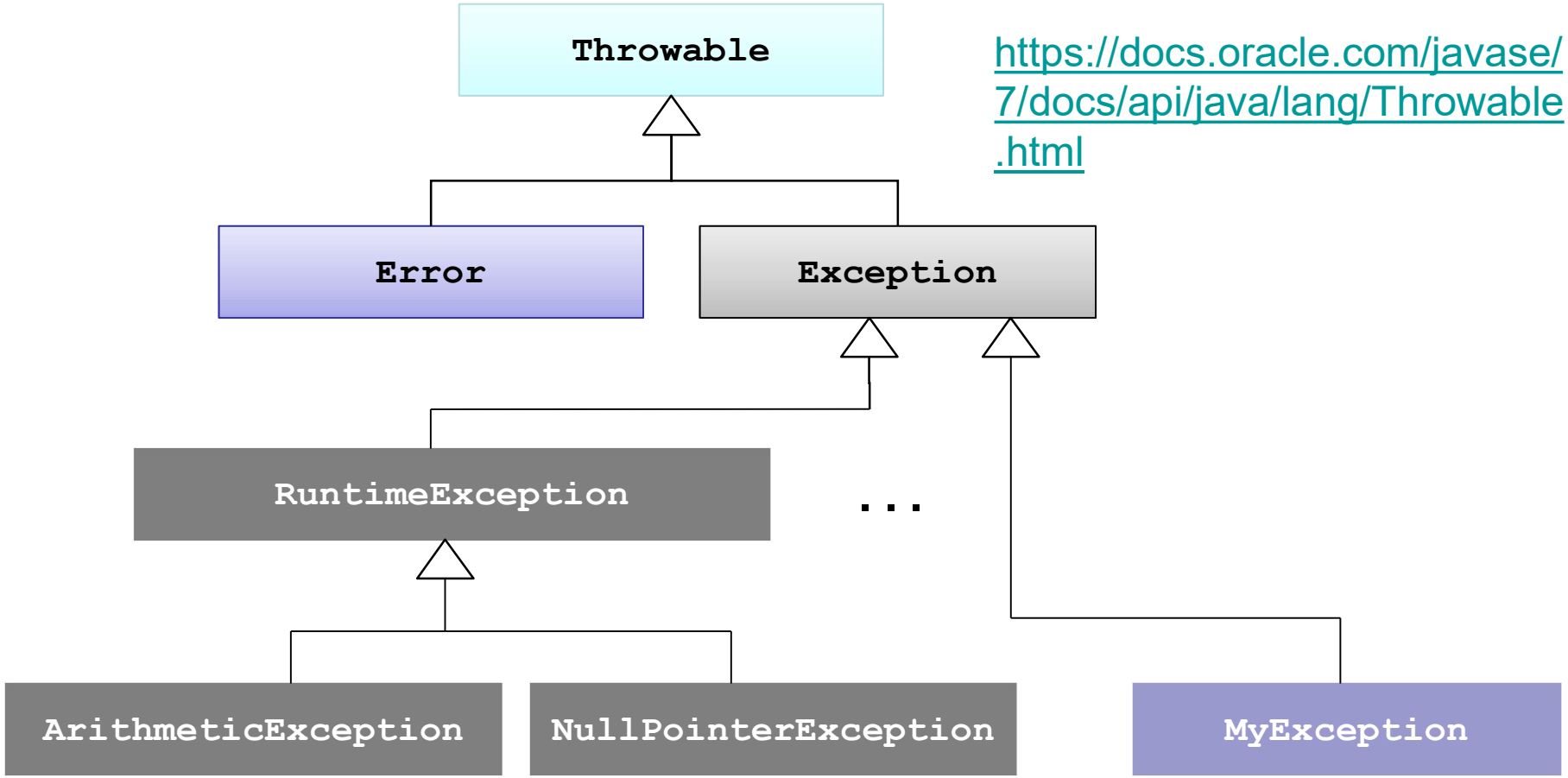
Chapter 11: Characters and Strings

Exception classes

Technische Hochschule Rosenheim

- So far
  - Objects of the type (class) `Exception` are thrown using `throw`, and caught with `catch`
  - No differentiation possible according to error type
- Improve the information content
  - Declaration of different error classes
  - All compatible with (i.e. derived from the class) `Throwable`
  - Response to errors in `catch` differentiated by type of exception

Exception classes (`java.lang`)



https://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html

Exception classes (`java.lang`)

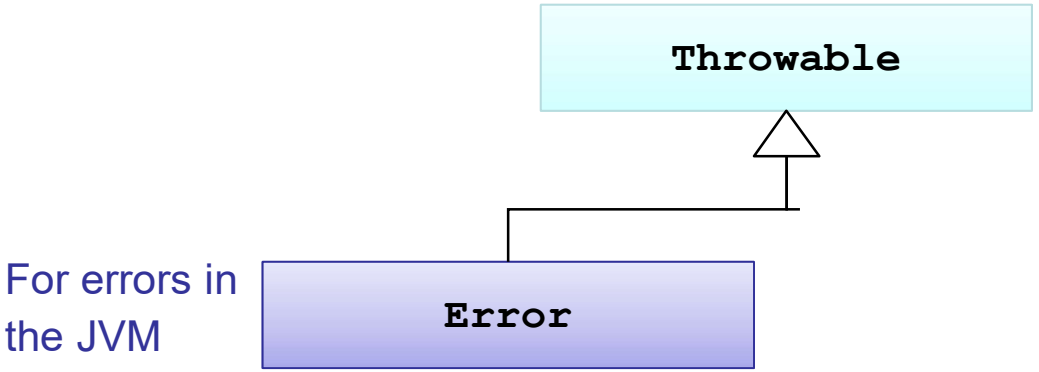
Technische Hochschule Rosenheim

Base class - not intended for direct use

- Exceptions in Java are implemented based on objects
- Information about exception situations is encapsulated in objects
- " ... The Throwable class is the superclass of all errors and exceptions in the Java language ... "
- Throwable is not an abstract class, instances can be created normally (various constructors)
- Throwable contains stack trace log information, about what the programme did last (e.g. which method calls, in which order --> programme logic)
- printStackTrace() allows the output of this log info --> we can see where the error occurred
- Direct subclasses are Error and Exception

Exception classes (`java.lang`)

Technische Hochschule Rosenheim



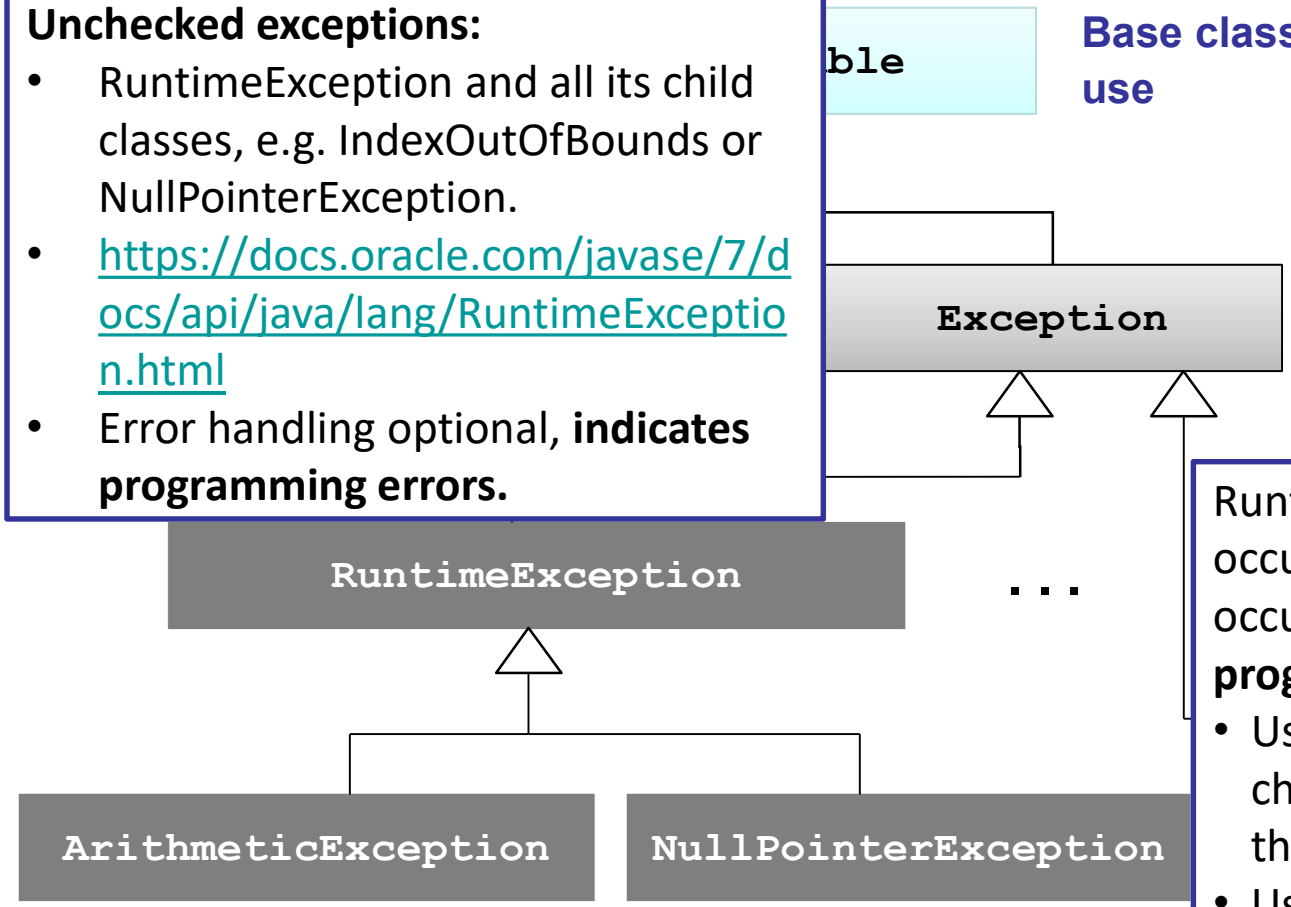
- Error: indicates serious errors, e.g. JVM crashes, Java internal errors. These are errors that cannot be fixed during execution.
- Using the Error class is about providing a methodology when programming for the Java framework.
- These serious errors have the **Error** class as a common basis.
- They are not checked by the compiler and do not (and should not) have to be caught!
- Examples:
  - NoClassDefFoundError: thrown when a class is not found
  - StackOverflowError: thrown when the stack has overflowed
  - OutOfMemoryError: thrown when there is no space left on the heap

Exception classes (`java.lang`)

Technische Hochschule Rosenheim

Unchecked exceptions:

- RuntimeException and all its child classes, e.g. IndexOutOfBounds or NullPointerException.
- https://docs.oracle.com/javase/7/docs/api/java/lang/RuntimeException.html
- Error handling optional, **indicates programming errors.**

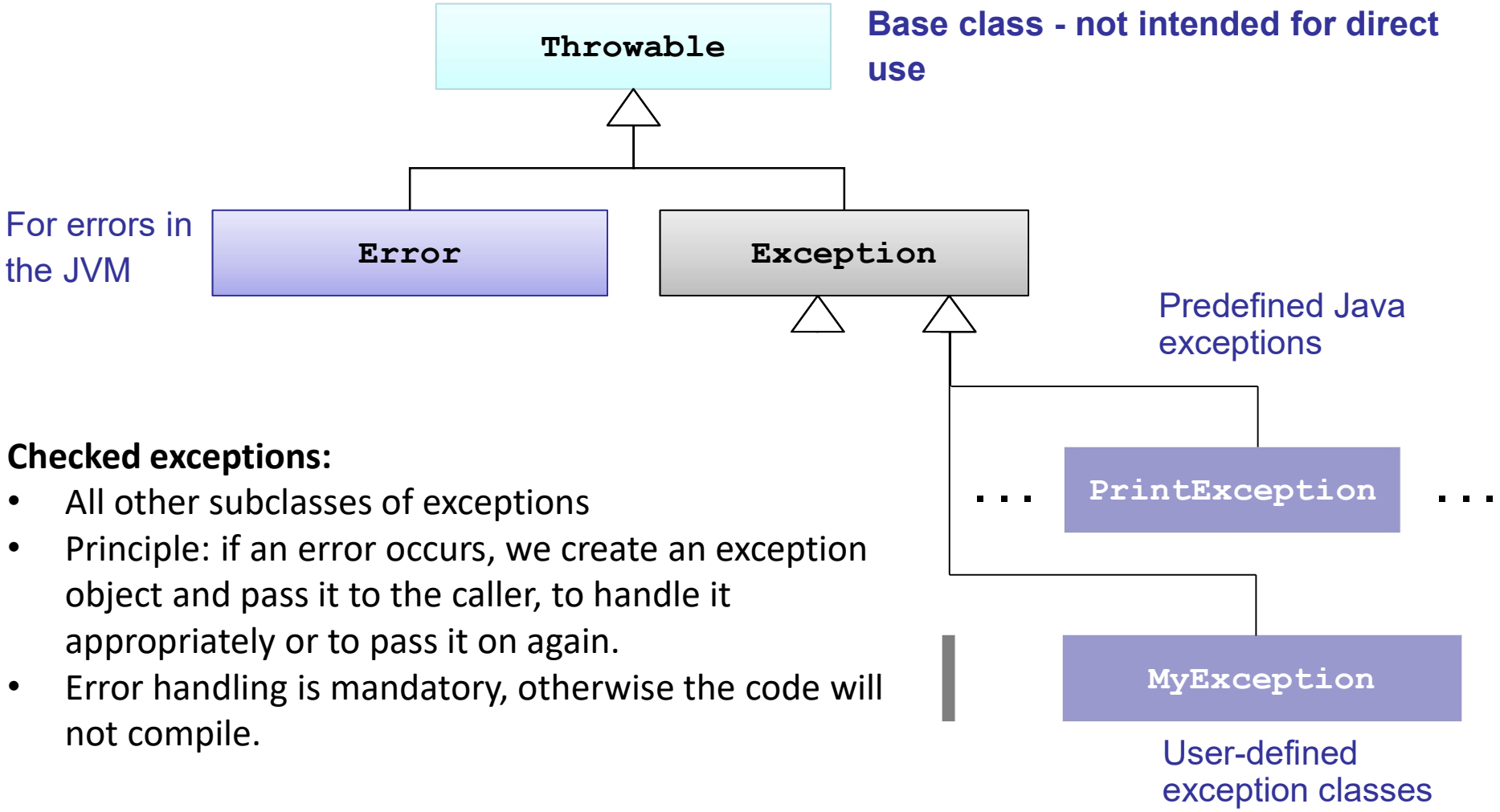


RuntimeException should not occur at all => try to prevent its occurrence by **defensive programming**

- Use index < array.length to check whether the array has the correct size or not
- Use name != null to check whether the corresponding variable is initialised or not

Many exception classes are predefined

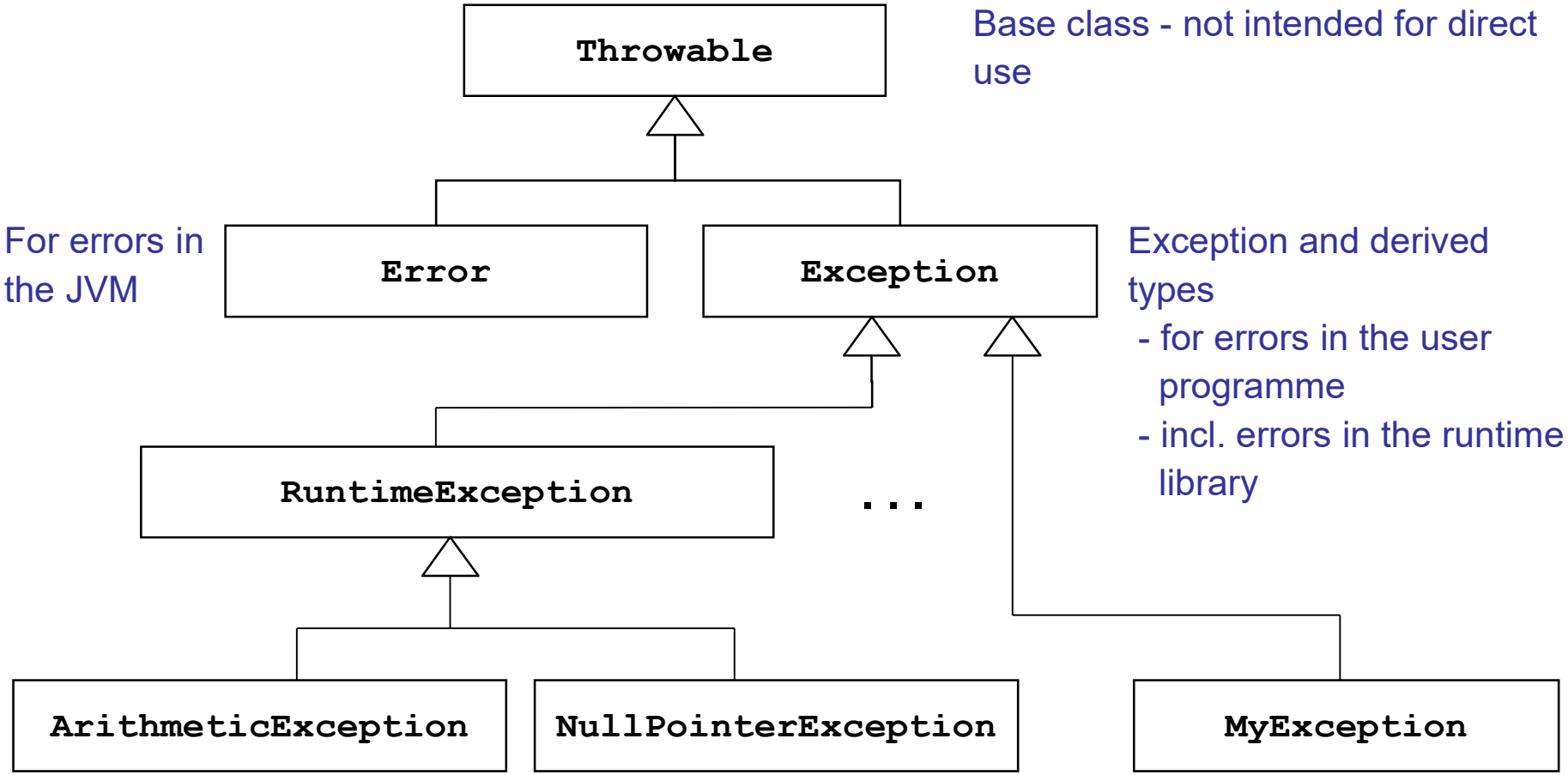
Exception classes (`java.lang`)



Checked exceptions:

- All other subclasses of exceptions
- Principle: if an error occurs, we create an exception object and pass it to the caller, to handle it appropriately or to pass it on again.
- Error handling is mandatory, otherwise the code will not compile.

Predefined exception classes (`java.lang`)



Predefined exception classes (2)

Technische Hochschule Rosenheim

- In some cases, we can find suitable classes and use them (specific cause of error should be classified!)

- Example:

```
public String clip(String s) throws Exception{
  if (s == null)
      throw new NullPointerException("no string");
  if (s.length() < 2)
      throw new IllegalArgumentException("short string");

  return s.substring(1, s.length() - 1);
}
```

User-defined exception classes (1)

Technische Hochschule Rosenheim

- Problem: predefined exception classes are not always suitable
- Remedy: define your own exception classes
- For simple cases: derive empty exception class from `Exception`
  - Class has empty body

```
class StringClipException extends Exception {}
```

- Slightly more information content
  - Declare constructor with `String` parameters for error message

```
class StringClipException extends Exception {
  StringClipException () {}
  StringClipException (String message) {
    super(message);
  }
}
```

Exercise – User-defined exception classes

Technische Hochschule Rosenheim

- Live exercise
  - Complete Task 3a on the live exercises sheet “Exceptions”
  - You have 5 minutes.

User-defined exception classes (2)

Technische Hochschule Rosenheim

- Example: use of the customised exception class

```
String clip (String s) throws Exception {
  if (s == null)
    throw new StringClipException("no string");

  if (s.length() < 2)
    throw new StringClipException("string too short");

  return s.substring (1, s.length() - 1);
}
```

Exception signatures (1)

Technische Hochschule Rosenheim

- Exception signature in the method header shows callers which exceptions the method could throw
- List of exceptions can be specified

- Syntax:

```
returntype methodname(parameterlist)
    throws exceptiontype1, exceptiontype2, …
```

- Information as specific as possible for exception signature
- Compiler checks correctness: (checked exceptions)
  - Missing, incorrect, redundant exceptions in the signature are recognised and rejected by the compiler

Exception signatures (2)

Technische Hochschule Rosenheim

- The previous example may be correct, but it is not skilful
  - Caller must expect any `Exception`
  - However, only `StringClipException` can occur

```
String clip (String s) throws StringClipException {
  if (s == null)
    throw new StringClipException ("no string");

  if (s.length() < 2)
    throw new StringClipException ("string too short");

  return s.substring (1, s.length() - 1);
}
```

Important note: information as specific as possible is better! This means that the exception signature should be specified as accurately as possible.

Exercise – User-defined exception classes

Technische Hochschule Rosenheim

- Live exercise
  - Complete Task 3b on the live exercises sheet “Exceptions”
  - You have 5 minutes.

Exception signatures (3)

Technische Hochschule Rosenheim

- Passing on of exceptions
  - A method which is called in a `try` block often does not produce exceptions itself, but calls methods that throw exceptions. Example:

```
String tripleClip (String s) {
  return clip(clip(clip(s)));
}
```

  - Subordinate `clip` calls can trigger exceptions
  - `tripleClip` must "take responsibility" towards users

➔ In its signature, the method must

  - identify the types of self-triggered exceptions and also
  - those of all the methods called.

```
String tripleClip (String s) throws StringClipException {
  return clip(clip(clip(s)));
}
```

- Compiler checks completeness of the exception signature

Special role of `RuntimeException` (1)

Technische Hochschule Rosenheim

- Example:
  - ```
    Rational b = null;
    b.reduce();                    // NullPointerException
    ```
  - JVM throws `NullPointerException` if object is missing during access
- Problem
  - `NullPointerException` possible with every access to data elements and every method call
  - Should be in the signature of almost every method from the compiler's perspective
  - Cumbersome, impractical, difficult to read
- Compromise
  - Special `RuntimeException` type
  - Does not have to be specified in the method signature
    ```

Special role of `RuntimeException` (2)

Technische Hochschule Rosenheim

- Use of `RuntimeException`
  - Reserved for omnipresent errors (standard runtime exceptions – unchecked exceptions)
  - No `throws` clause required
  - Mostly in connection with certain language constructs
  - Occurrence of `RuntimeException` usually sign of weaknesses in programme logic
  - Better: Ensure that classic cases of error are queried in case differentiation

- Examples: `ClassCastException, NullPointerException, IndexOutOfBoundsException, ArithmeticException,…`

Special role of `Error`

Technische Hochschule Rosenheim

- Meaning
  - `Error` and derived types are intended for errors of the JVM
  - User programme should not explicitly trigger `Error`
  - No meaningful reaction to `Error` is usually possible
  - Therefore, the user programme should not catch and handle `Error`
  - `Error` does not need to be declared in exception signature (unchecked exceptions)
- Examples:
  - `OutOfMemoryError`: JVM used up all available memory
  - `ClassFormatError`: Attempt to load defective bytecode
  - `VirtualMachineError`: Internal error in the JVM

Chapter 11: Characters and Strings

11.1 Motivation, definition of terms and procedure

11.2 Exception classes

11.3 Other implementation aspects

Handling exceptions (1)

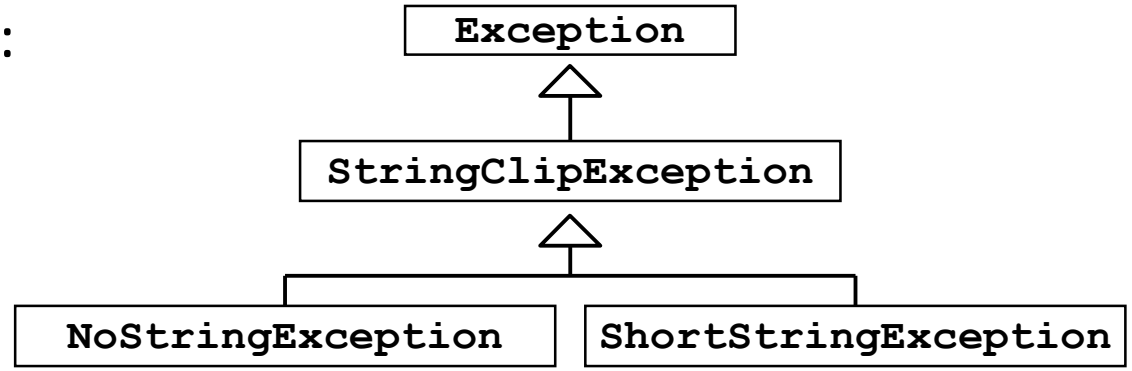
Technische Hochschule Rosenheim

- Multiple `catch` clauses
  - Method can trigger exceptions of different types
  - Catch in several subsequent `catch` blocks
  - In case of an error:
    `catch` blocks are compared sequentially with the exception
  - The first compatible ("matching") `catch` block applies
  - Subsequent (possibly also matching) `catch` blocks are ignored

Handling exceptions (2)

Technische Hochschule Rosenheim

- Example:




```
String clip (String s) throws NoStringException, ShortStringException {
  if (s == null)
    throw new NoStringException();

  if (s.length() < 2)
    throw new ShortStringException();

  return s.substring(1, s.length() - 1);
}
```

Handling exceptions (3)

Technische Hochschule Rosenheim

- Example: Application of `clip` method

```
public static void main (String[] args) {
  String s = "Test string";

  try {
    System.out.println(clip(s));
  } catch (NoStringException ex) {
      System.out.println("no string to clip");
  } catch (ShortStringException ex) {
      System.out.println("cannot clip string");
  }
}
```

  - If `s == null`: `NoStringException`
    - First `catch` block matches, output of `"no string to clip"`
    - Second `catch` block also matches, but is no longer taken into account
  - If `s = "a"`: `ShortStringException`
    - First `catch` block does not match
    - Second `catch` block matches, output of `"cannot clip string"`

Exercise – User-defined exception classes

Technische Hochschule Rosenheim

- Live exercise
  - Complete Task 3c on the live exercises sheet “Exceptions”
  - You have 5 minutes.

Handling exceptions (4)

Technische Hochschule Rosenheim

- Guidelines for use:
  - Specific types of exception to the front, more general types of exception to the back
  - Avoid `catch (Exception ex)`
    - Catches too many exceptions
    - In particular, exceptions of the `RuntimeException` type
- Missing catch block
  - This means that the exception thrown is not caught anywhere
  - Exception reaches JVM
  - JVM stops the programme execution

Summary of checked exceptions

Technische Hochschule Rosenheim

- Checked exceptions are either
  - handled locally
    (within the method in which they occur) or
  - they are part of the method signature
    (`throws` clause declares the checked exceptions that a method can trigger)
- The caller of a method has three options:
  (1) Catch and handle the exception
  (2) Catch the exception and throw a new exception
  (3) Declare the exception itself in a `throws` clause, thereby passing the exception to the own caller
- Idea:
  - Signature clarifies all that can happen
  - Callers are forced to react