**Programare Avansata pe Obiecte  
Laborator 6**

**Pahontu Bogdan-Ionut**

**E-mail:** [**pahontubogdan@gmail.com**](mailto:pahontubogdan@gmail.com)

Contents

[1. Interfaces (Part 2) 3](#_Toc5049672)

[1.1. Interfaces from Java 7 to Java 9 3](#_Toc5049673)

[2. Exceptions 4](#_Toc5049674)

[2.1. Understanding exception types 4](#_Toc5049675)

[2.2. Using a try statement 5](#_Toc5049676)

[2.3. throw VS throws 6](#_Toc5049677)

[2.4. Custom exceptions 6](#_Toc5049678)

[2.5. Using multi-catch 7](#_Toc5049679)

[2.6. Using try-with-resources 7](#_Toc5049680)

[2.7. Autoclosable 8](#_Toc5049681)

[3. Tasks 8](#_Toc5049682)

# Interfaces (Part 2)

## Interfaces from Java 7 to Java 9

* In Java 7 or earlier versions, an interface can have only two things
  + Constant variables;
  + Abstract methods;
* In Java 8 new features were added to an interface (Default methods and Static methods feature), so now we have the following:
  + Constant variables;
  + Abstract methods;
  + Default methods;
  + Static methods;
* In Java 9 new features were added, private methods and private static method, so the following are available from this version:
  + Constant variables;
  + Abstract methods;
  + Default methods;
  + Static methods;
  + Private methods;
  + Private static methods;
* Rules to define private/ private static methods:
  + Private interface method cannot be abstract
  + Private method can be used only inside interface and other static and non-static interface methods.
  + Private non-static methods cannot be used inside private static methods.
* In order to run this examples you need to have installed Java 9. You can download Java 9 from the following link: <https://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase9-3934878.html>, but an oracle account is required.
* For a concrete example visit: <https://www.geeksforgeeks.org/private-methods-java-9-interfaces/>

# Exceptions

## Understanding exception types

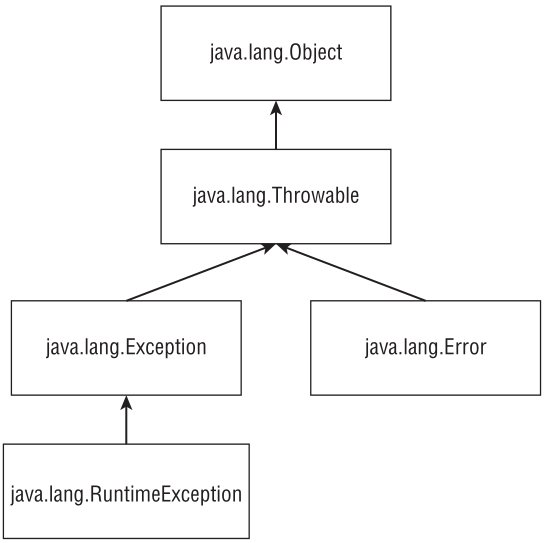


Figure 1 – Understanding exceptions

* **Error:**  means something went so horribly wrong that your program should not attempt to recover from it. For example, the disk drive “disappeared.”
* **Runtime exceptions:** exceptions tend to be unexpected but not necessarily fatal. Ex: accessing an invalid array index is unexpected, NullPointerException . Runtime exceptions are also known as **unchecked** **exceptions**.
* **Checked exceptions**: tend to be more anticipated—for example, trying to read a file that doesn’t exist.

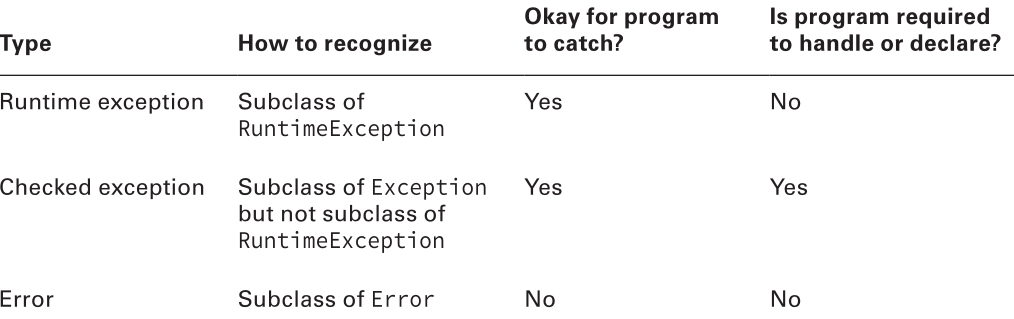


Figure 2 – Exceptions/Error handling

Please check “UncheckedExceptionsExample.java” and “CheckedExceptionExample.java”

* Common runtime exceptions include the following:
  + **ArithmeticException** Thrown by the JVM when code attempts to divide by zero;
  + **ArrayIndexOutOfBoundsException** Thrown by the JVM when code uses an illegal index to access an array;
  + **ClassCastException** Thrown by the JVM when an attempt is made to cast an object to a subclass of which it is not an instance;
  + **IllegalArgumentException** Thrown by the programmer to indicate that a method has been passed an illegal or inappropriate argument;
  + **NullPointerException** Thrown by the JVM when there is a null reference where an object is required
  + **NumberFormatException** Thrown by the programmer when an attempt is made to convert a string to a numeric type but the string doesn’t have an appropriate format (wrappers: Integer etc);

Please check: RuntimeExceptionsEx.java file.

* Errors classes extend **Error** class;
  + **ExceptionInInitializerError** Thrown by the JVM when a static initializer throws an exception and doesn’t handle it;
  + **StackOverflowError** Thrown by the JVM when a method calls itself too many times (this is called infinite recursion because the method typically calls itself without end)
  + **NoClassDefFoundError** Thrown by the JVM when a class that the code uses is available at compile time but not runtime;

## Using a try statement

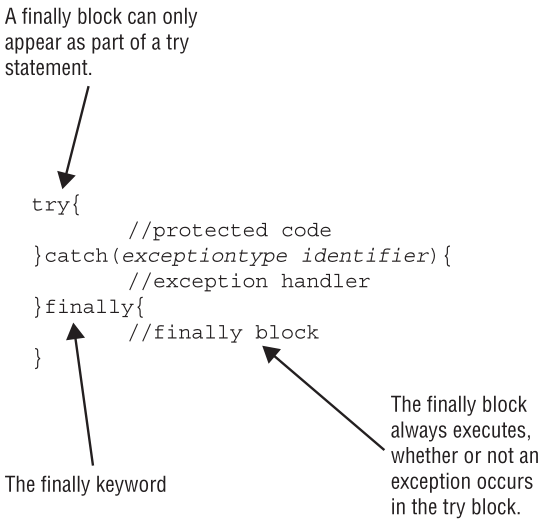


Figure 3 – Try/catch example

* Try statements to separate the logic that might throw an exception from the logic to handle that exception.
* It can include one or more catch clauses to handle the exceptions that are thrown.
* It can also include a finally clause, which runs regardless of whether an exception is thrown.
* A try statement is required to have either or both of the catch and finally clauses

Please check: RuntimeExceptionsEx.java file.

## throw VS throws

* **throw** means an exception is actually being thrown
* **throws** indicate that the method merely has the potential to throw that exception.
* The following example uses both:

public String getDataFromDatabase() **throws** SQLException {

**throw** new UnsupportedOperationException();

}

## Custom exceptions

public class CannotSwimException extends Exception {

public CannotSwimException() {

super();

}

public CannotSwimException(Exception e) {

super(e);

}

public CannotSwimException(String message) {

super(message);

}

}

* The first constructor is the default constructor with no parameters. The second constructor shows how to wrap another exception inside yours. The third constructor shows how to pass a custom error message.

## Using multi-catch

* Syntax of multi-catch is like a regular catch clause, except two or more exception types are specified separated by a pipe;
* The pipe is also used as the “or” operator, making it easy to remember that you can use either/or of the exception types;
* Notice how there is only one variable name in the catch clause. Java is saying that the variable named e can be of type Exception1 or Exception2
* Java intends **multi-catch** to be used **for exceptions that aren’t related**, and it prevents you from specifying redundant types in a multi-catch

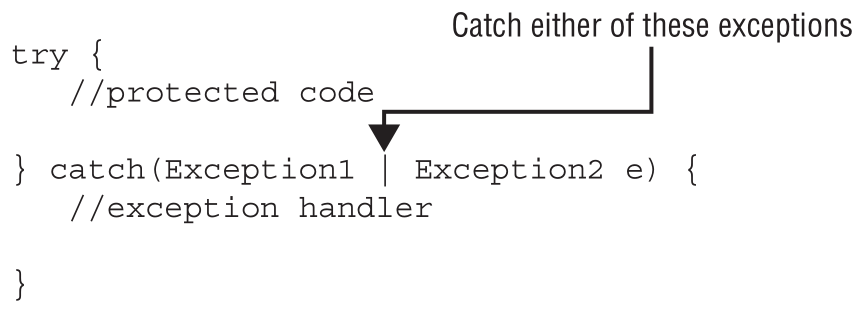


Figure 4 – Try/catch example

## Using try-with-resources

* One or more resources can be opened in the try clause.
* Parentheses are used to list those resources and semicolons are used to separate the declarations

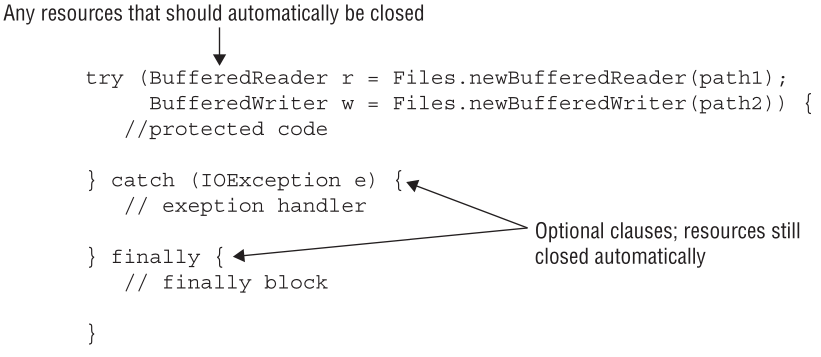


Figure 4 – Try with resources example

## Autoclosable

* You can’t just put any random class in a try-with-resources statement. Java commits to closing automatically any resources opened in the try clause;
* All the resources defined in the try-with-resources statement need to implement the AutoCloseable interface
* When multiple exceptions are thrown, all but the first are called **suppressed exceptions**
* The first exception thrown is called a **primary exception** -> all other exceptions thrown in the try or close() sections are tacked to the primary exception and are called suppressed exceptions

Please check “UncheckedExceptionsExample.java” and “CheckedExceptionExample.java”

# Tasks

**Task 1:**

1. Download and install JDK 12:

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>

**Task 2:**

1. Create a new package named tasks in lab6 project;
2. Create a packet named model. Inside this packet create: Authenticable, AuthenticableDB. AuthenticableLDAP, User;
3. Implement three interfaces, all with a method with a line like:

System.out.println(< Curent time> : <className> -> <metodName>)

Authenticable

default void performAuthentication() throws Exception

AuthenticableLDAP extends Authenticable

private void performLDAPAuth()

default void performAuthentication() throws Exception

t

AuthenticableDB extends Authenticable

private void performDBAuth()

default void performAuthentication()

throws Exception

1. The performAuthentication () method will call performLDAPAuth or performDBAuth depending on the interface in which we implement.
2. Implement the User class that implements both AuthenticableLDAP and AuthenticableDB interfaces. Override the performAuthentication method to avoid compilation errors;
3. Search on internet about Java enum;
4. The user class will have an AuthType internal enum with two possible DB and LDAP values.
5. User's constructor will contain an AuthType enum parameter.

User implements AuthenticableDB, AuthenticableLDAP

String name

String hashPassword

AuthType authType

public void performAuthentication()

throws Exception

1. An exception package will be created as a package under tasks.
2. A custom custom exception class, UserAuthException, will be implemented in the already created package. There is a constructor with the String msg parameter that will be sent to the parent class constructor.
3. The public void method performAuthentication () of the User class implements a switch that chooses, depending on the authType parameter, to call a parameter or performance-based parameter (DB or LDAP) method.
4. Test your code in a main method in a new class.