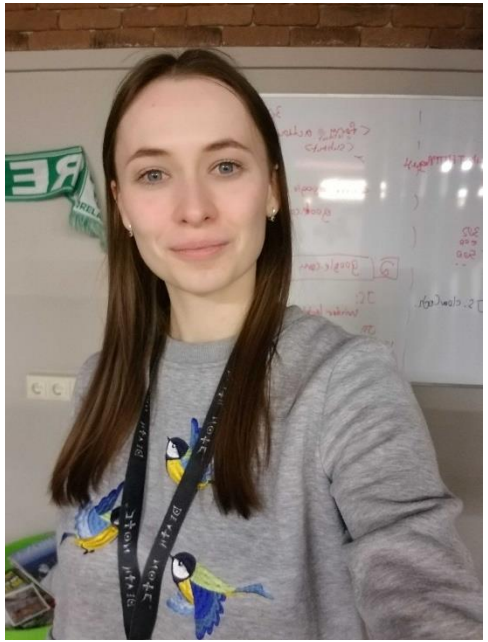
Java Training
Exceptions in java

Our Team



Palina Nosava, 7 years at EPAM, Software Engineer

Irina Savitskaya, 12+ years at EPAM, Lead Software Engineer

Aleh Drazdou, 13+ years at EPAM, Lead Software Engineer

Today's agenda

- What is an Exception?
- The Three Kinds of Exceptions
- How to Catch and Handle Exceptions?
- The Exceptions Thrown by a Method
- How to write your own exception classes?
- Exceptions and Method Overriding
- Exceptions in Constructors*

What is an Exception?

Throwing an exception

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

Throwing an exception is the process of creating an exception object and handing it to the runtime system.

Example

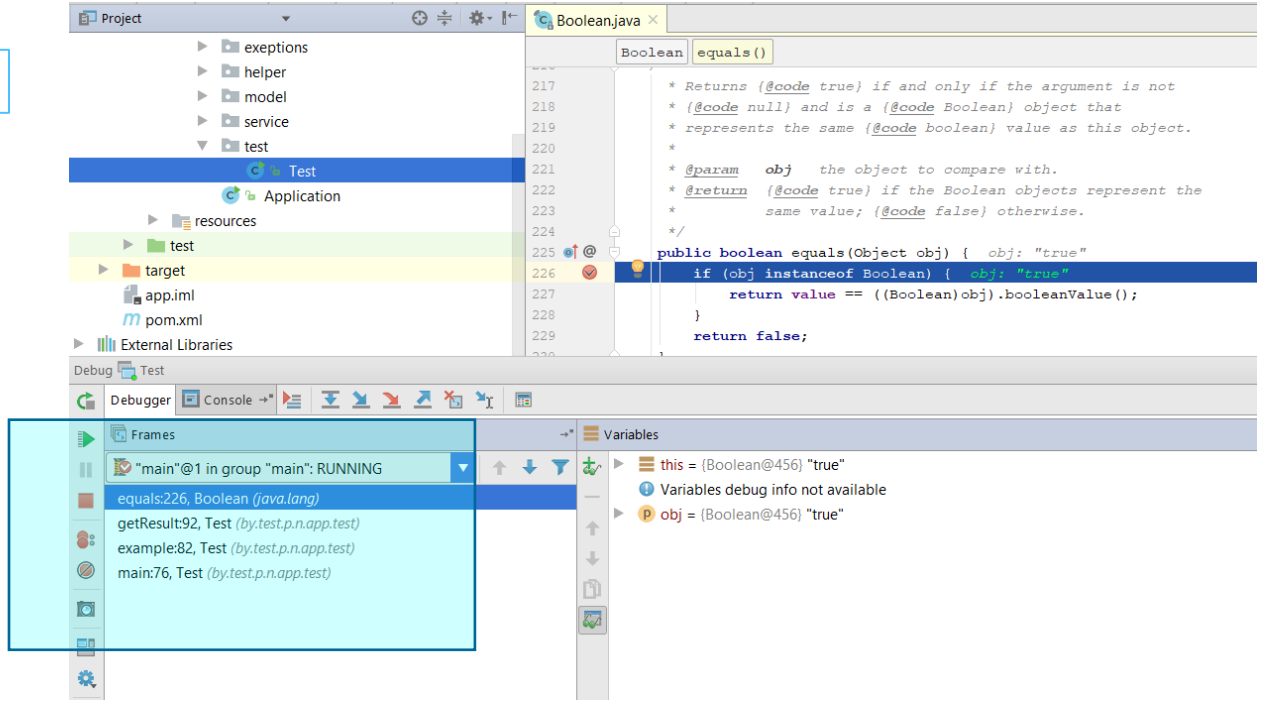
```
public static void main(String[] args) {
        Object o = null;
        o.toString();
}
```

Exception in thread "main"
java.lang.NullPointerException

The call stack

The call stack is the ordered list of methods that had been called to get to the method where the error occurred.

Example



A Stack trace

A Stack trace is an information on the execution history of the current thread and lists the names of the classes and methods that were called at the point when the exception occurred.

Example

```
java.sql.SQLException: [Amazon](500150) Error setting/closing connection: UnknownHostException.
    at com.amazon.redshift.client.PGClient.connect(Unknown Source)
    at com.amazon.redshift.client.PGClient.<init>(Unknown Source)
    at com.amazon.redshift.core.PGJDBCConnection.connect(Unknown Source)
    at com.amazon.jdbc.common.BaseConnectionFactory.doConnect(Unknown Source)
    at com.amazon.jdbc.common.AbstractDriver.connect(Unknown Source)
    at com.zaxxer.hikari.util.DriverDataSource.getConnection(DriverDataSource.java:136)
    at com.zaxxer.hikari.pool.PoolBase.newConnection(PoolBase.java:369)
    at com.zaxxer.hikari.pool.PoolBase.newPoolEntry(PoolBase.java:198)
    at com.zaxxer.hikari.pool.HikariPool.createPoolEntry(HikariPool.java:467)
    at com.zaxxer.hikari.pool.HikariPool.access$100(HikariPool.java:71)
    at com.zaxxer.hikari.pool.HikariPool$PoolEntryCreator.call(HikariPool.java:706)
    at com.zaxxer.hikari.pool.HikariPool$PoolEntryCreator.call(HikariPool.java:692)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
Caused by: com.amazon.support.exceptions.GeneralException: [Amazon](500150) Error setting/closing connection: UnknownHostException.
    ... 15 more
```

The Three Kinds of Exceptions

Checked, Unchecked Exception and Error

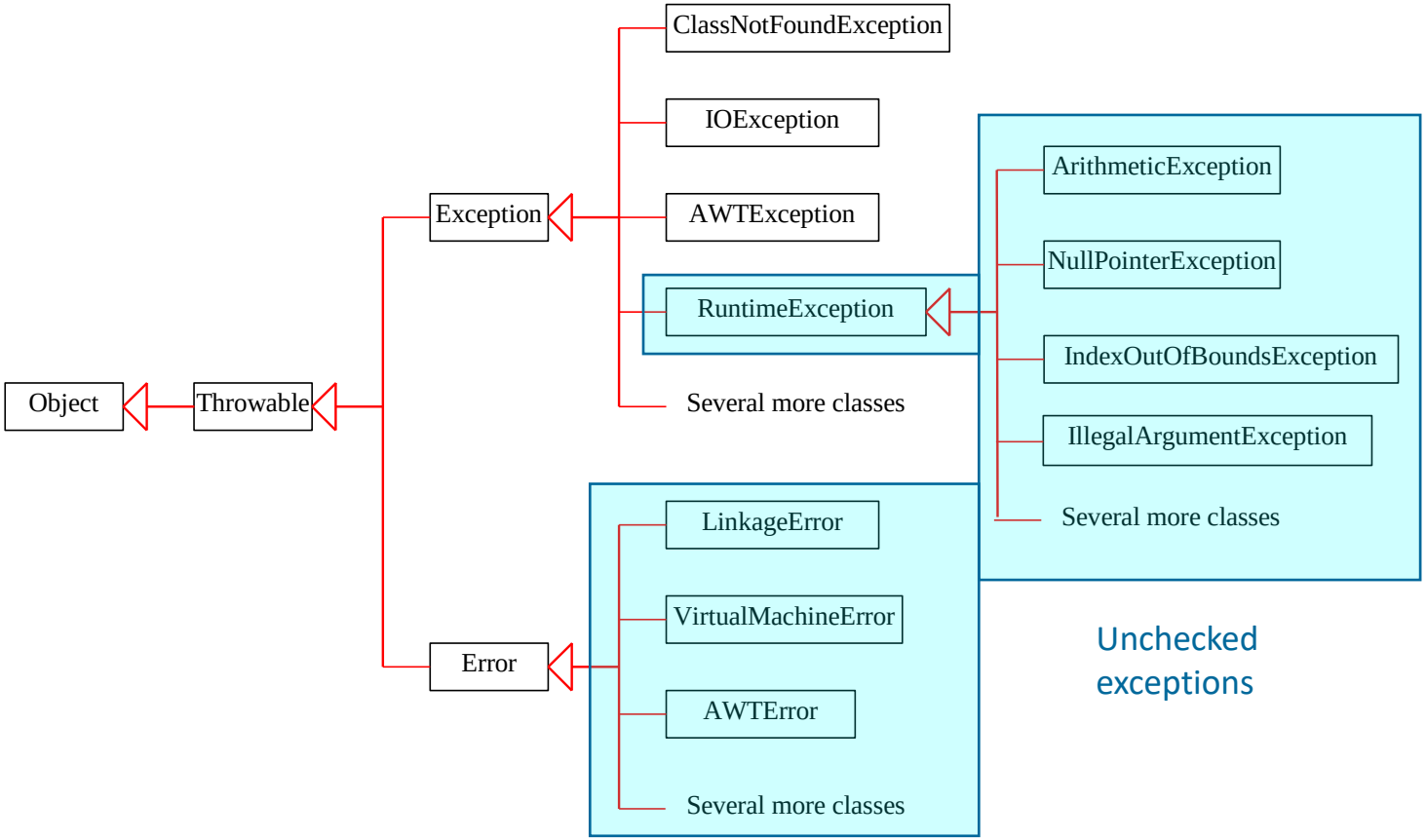
A Checked Exception is an exceptional condition that a well-written application should anticipate and recover from. If your code invokes a method which is defined to throw checked exception, your code MUST provide a catch handler. The compiler generates an error if the appropriate catch handler is not present.

An Error is an exceptional condition that is *external* to the application, and that the application usually cannot anticipate or recover from.

A Runtime Exception is an exceptional conditions that is *internal* to the application, and that the application usually cannot anticipate or recover from.

An Unchecked Exception is the exception indicated by Errors and Runtime Exceptions.

Exception Hierarchy



How to Catch and Handle Exceptions?

Try Catch Finally syntax

```
try {
        // statement that could throw an exception
}
catch (<exception type> e1) {
        // statements that handle the exception
}
catch (<exception type> e2) {//e2 higher in than e1
        // statements that handle the exception
}
finally {
        // release resources
}
//other statements
```

The Try statement is a block of code that might throw certain exceptions.

An exception object is an object that is created within the method and is handed off to the runtime system when an error occurs. The object, called, contains information about the error, including its type and the state of the program when the error occurred.

The Catch block is an exception handler that handles the type of exception indicated by its argument. It's body contains the code that is executed if and when the exception handler is invoked.

The Finally block is a block of code that always executes when the try block exits. Usually contains cleanup code.

Try Catch Finally syntax

```
try {

        // statement that could throw an exception

}
catch (<exception type> e1) {

        // statements that handle the exception

}
catch (<exception type> e2) { //e2 higher in hierarchy than e1

        // statements that handle the exception, same as on catch above

}
finally {

        // release resources
}

//other statements
```

Suppose that the handle logic should be the same for both exception types

Try Catch Finally syntax

```
try {
        // statement that could throw an exception
}
catch (<exception type 1>|<exception type 2> e) {
        // statements that handle the exception
}
finally {
        // release resources
}
//other statements
```

Exceptions can be combined in one catch statement

Try-with-resources syntax

The Try-with-resources statement is a try statement that declares one or more resources. It ensures that each resource is closed at the end of the statement. A try-with-resources statement can have catch and finally blocks just like an ordinary try statement. In a try-with-resources statement, any catch or finally block is run after the resources declared have been closed.

```
try (<Closeable_Type> variable = new <Closeable_Type>()) {
        // statement that could throw an exception
}
```

Any object that implements **java.lang.AutoCloseable or java.io.Closeable** can be used as a resource.

Example

```
static String readFirstLineFromFile(String path) throws IOException {
    try (BufferedReader br = new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
}
```

The Exceptions Thrown by a Method

Throws and Throw syntax

```
<access_modifier> <key_words> <return_type> <method_name> (<method variables>)
throws <exceptions_list> {

    // method body

    throw new <exception_type>();

    //return statement if needed
}
```

A Throws clause is a condition that lists the exceptions that can be thrown by the method. It is not mandatory to include unchecked exceptions to throws clause.

Example

```
public void writeList() throws IOException{

    // method body

    throw new IOException("Exception message");

}
```

A Throw statement is a code that creates a throwable object and uses 'throw' keyword to throw the exception. It is possible to throw the exception from anywhere.

How to write your own exception classes?

Best practices for custom exceptions

1. Prefer Specific Exceptions. The more specific the exception that you throw is, the better. Always try to find the class that fits best to your exceptional event and first look at existing Exceptions before creating your own.
2. Follow the naming convention. Append the string Exception to the names of all classes that inherit (directly or indirectly) from the Exception class.
3. Provide Javadoc comments for your exception class. The Javadoc should describe the general meaning of the exception and the situations in which it might occur. The goal is to help other developers to understand your API and to avoid common error scenarios.
4. Provide a constructor that sets the cause. You should implement at least one constructor that gets the causing Throwable as a parameter and sets it on the superclass.

Custom Exception syntax

```
<access_type> class <class_name> extends <parent_exception_class_name>{
        // constructors that call super constructors
}
```

Example

Use 'Exception' word in the name

Use appropriate Parent Exception class, based on the needs of your application

```
class InvalidAgeException extends Exception{

        InvalidAgeException(String s){
                super(s);
        }
}
```

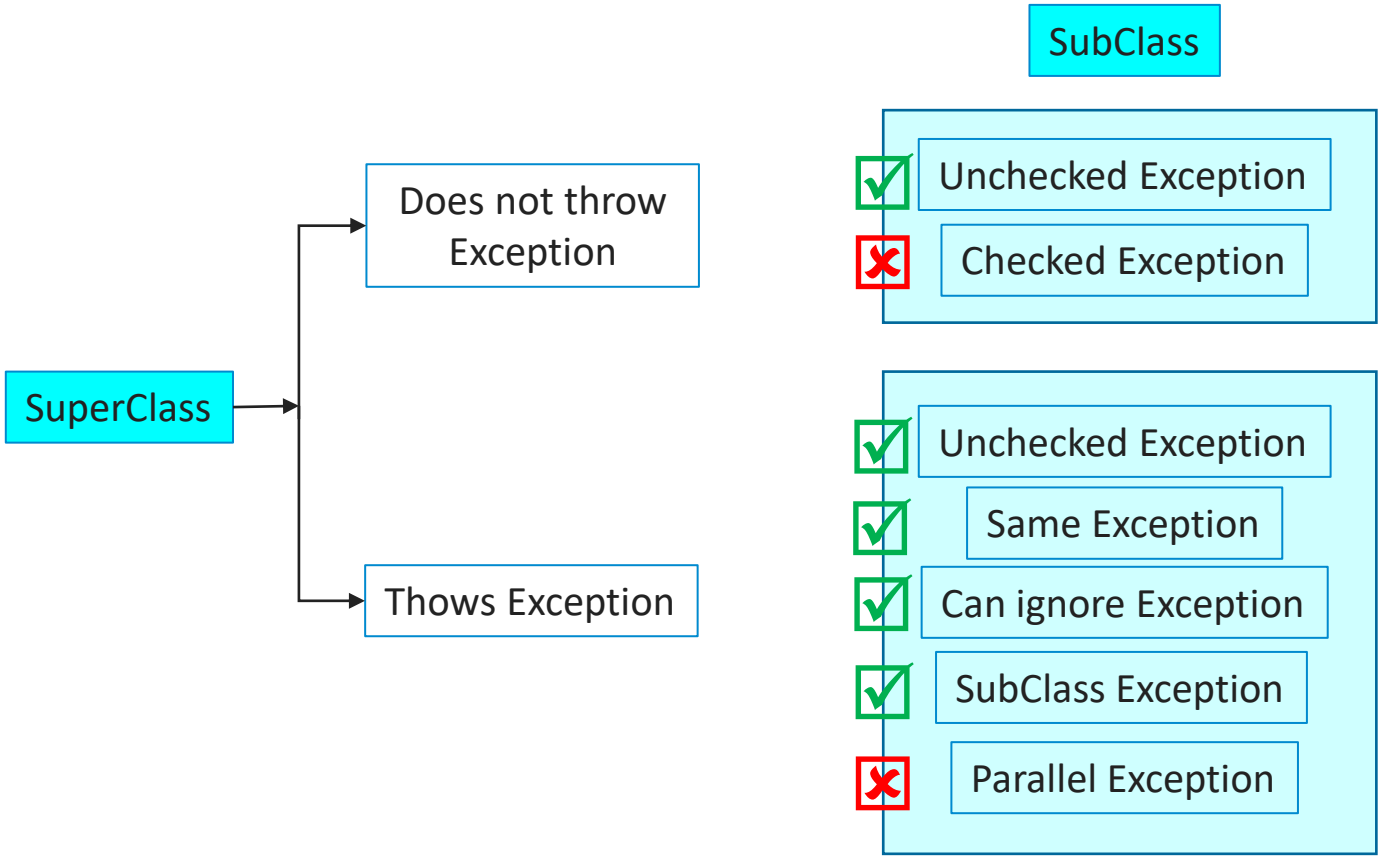
Use Parent's constructor

Exceptions and Method Overriding

Main conclusions

1. If SuperClass does not declare an exception, then the SubClass can only declare unchecked exceptions, but not the checked exceptions.
2. If SuperClass declares an exception, then the SubClass can only declare the child exceptions of the exception declared by the SuperClass, but not any other exception.
3. If SuperClass declares an exception, then the SubClass can declare without exception.

Exception Handling Rules in Overriding



Exceptions in Constructors*

1. Checked and unchecked exceptions can be thrown in constructor. In both cases, an object is actually allocated in the heap space, but a reference to it is not returned. The object remains in a partially initialized state until it gets garbage collected. It is risky since we may give access to an object in an invalid state.

Homework

- Solve problems: https://www.hackerrank.com/domains/java?filters%5Bsubdomains%5D%5B%5D=handling-exceptions
- Self Practice :
  - Is it possible to create the whole application based on exceptions? Think about it.
  - Use inheritance and override concepts. Create a program that will work with ‘throw’ and ‘throws’ key words in overridden methods. Explore behavior.
  - * Create a program that will work with ‘throw’ and ‘throws’ key words in constructors. Explore behavior.

Recommended materials

- Java Docs
- Code Complete, McConnell
- Effective Java, Bloch
- Building Maintainable Software (java edition), Visser
- Thinking in Java, Eckel

Thanks