Object-oriented programming

Lecture №9

Exception

PhD, Alexander Vlasov

Question?

- *Typing*
  - static / dynamic typing
  - strong / weak typing
  - explicit / implicit typing
  - Duck typing
- *Polymorphism*
  - *Ad hoc polymorphism*
  - *Parametric polymorphism*
  - *Subtype polymorphism*
- Java
  - Final
  - Static
  - (Anonymous) Inner Classes
  - Nested class

Exception - What is it and why do I care?

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

- Exception is an Object
- Exception class must be descendent of Throwable.

Exception - What is it and why do I care?

By using exceptions to manage errors, Java programs have the following advantages over traditional error management techniques:

1: Separating Error Handling Code from "Regular" Code
2: Propagating Errors Up the Call Stack
3: Grouping Error Types and Error Differentiation

Separating Error Handling Code from "Regular" Code

```
readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

Separating Error Handling Code from "Regular" Code

```
errorCodeType readFile {
    initialize errorCode = 0;
    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        } else {
            errorCode = errorCode and -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```

Separating Error Handling Code from "Regular" Code

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

Propagating Errors Up the Call Stack

```
method1 {
    try {
        call method2;
    } catch (exception) {
        doErrorProcessing;
    }
}
method2 throws exception {
    call method3;
}
method3 throws exception {
    throw new Exception("Error");
}
```

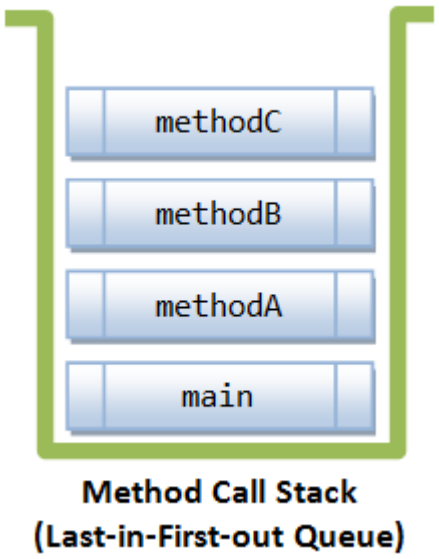
Example

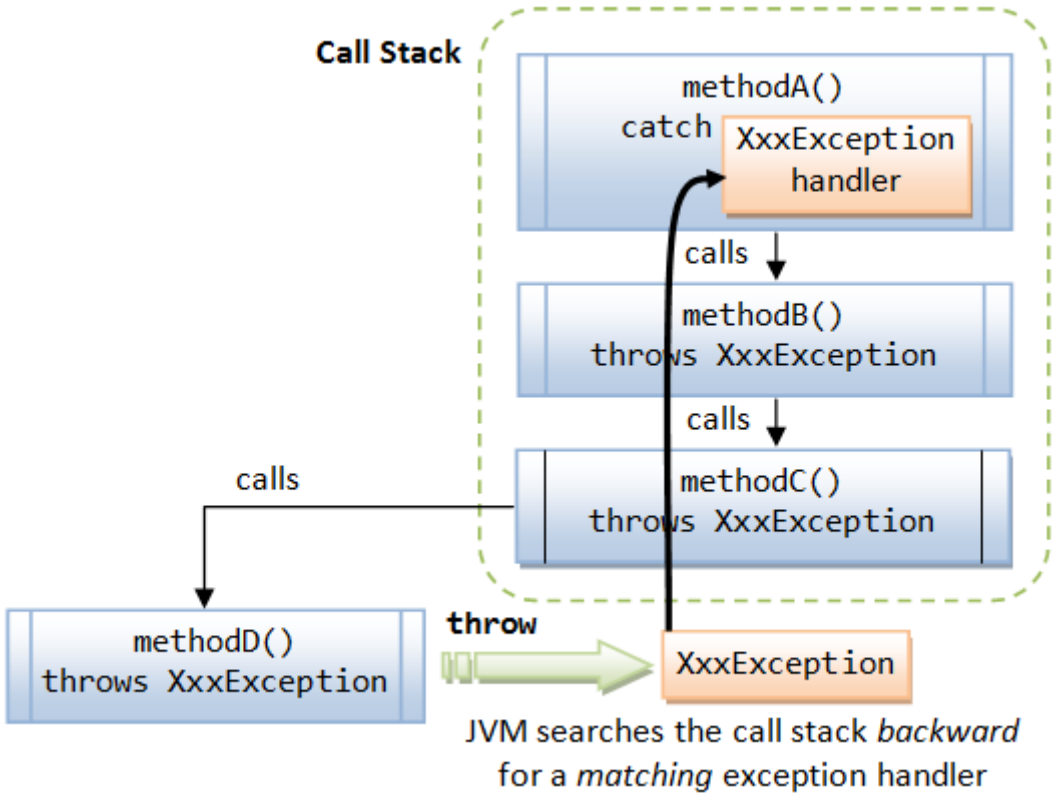
```
public class MethodCallStackDemo {
  public static void main(String[] args) {
    System.out.println("Enter main()");
    methodA();
    System.out.println("Exit main()");
  }
  public static void methodA() {
    System.out.println("Enter methodA()");
    methodB();
    System.out.println("Exit methodA()");
  }
  public static void methodB() {
    System.out.println("Enter methodB()");
    methodC();
    System.out.println("Exit methodB()");
  }
  public static void methodC() {
    System.out.println("Enter methodC()");
    System.out.println(1 / 0);  // divide-by-0 triggers an ArithmeticException
    System.out.println("Exit methodC()");
  }
}
```

```
Enter main()
Enter methodA()
Enter methodB()
Enter methodC()
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at MethodCallStackDemo.methodC(MethodCallStackDemo.java:22)
        at MethodCallStackDemo.methodB(MethodCallStackDemo.java:16)
        at MethodCallStackDemo.methodA(MethodCallStackDemo.java:10)
        at MethodCallStackDemo.main(MethodCallStackDemo.java:4)
```

```
Enter main()
Enter methodA()
Enter methodB()
Enter methodC()
Exit methodC()
Exit methodB()
Exit methodA()
Exit main()
```

Example

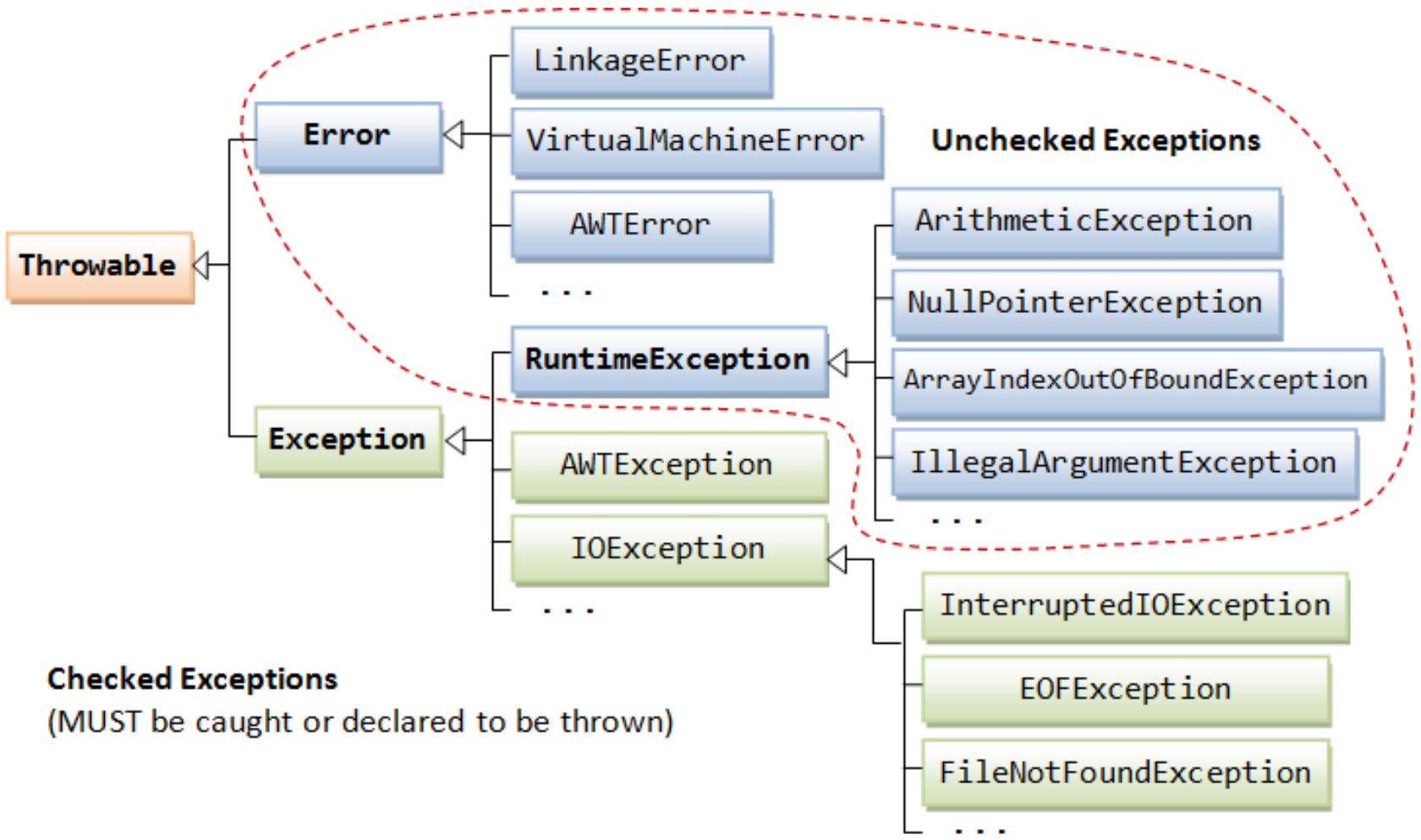

methodC
methodB
methodA
main
Method Call Stack
(Last-in-First-out Queue)


Call Stack
methodA()
catch
XxxException
handler
calls
methodB()
throws XxxException
calls
methodC()
throws XxxException
calls
methodD()
throws XxxException
throw
XxxException
JVM searches the call stack backward
for a matching exception handler

try – catch – finally

```
try {
   // main logic, uses methods that may throw Exceptions
   ......
} catch (Exception1 ex) {
   // error handler for Exception1
   ......
} catch (Exception2 ex) {
   // error handler for Exception1
   ......
} finally {   // finally is optional
   // clean up codes, always executed regardless of exceptions
   ......
}
```

Checked and unchecked exceptions



Constructs *Throwable*

Throwable() - constructs a new throwable with null as its detail message.

Throwable(String message) - constructs a new throwable with the specified detail message.

Throwable(String message, Throwable cause) - constructs a new throwable with the specified detail message and cause.

protected Throwable(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace) - constructs a new throwable with the specified detail message, cause, suppression enabled or disabled, and writable stack trace enabled or disabled.

Throwable(Throwable cause) - constructs a new throwable with the specified cause and a detail message of (cause==null ? null : cause.toString()) (which typically contains the class and detail message of cause).

Methods *Throwable*

void addSuppressed(Throwable exception)
Appends the specified exception to the exceptions that were suppressed in order to deliver this exception.
Throwable fillInStackTrace()
Fills in the execution stack trace.
Throwable getCause()
Returns the cause of this throwable or null if the cause is nonexistent or unknown.
String getLocalizedMessage()
Creates a localized description of this throwable.
String getMessage()
Returns the detail message string of this throwable.
StackTraceElement[] getStackTrace()
Provides programmatic access to the stack trace information printed by printStackTrace().
Throwable[] getSuppressed()
Returns an array containing all of the exceptions that were suppressed, typically by the try-with-resources statement, in order to deliver this exception.
Throwable initCause(Throwable cause)
Initializes the cause of this throwable to the specified value.
void printStackTrace()
Prints this throwable and its backtrace to the standard error stream.
void printStackTrace(PrintStream s)
Prints this throwable and its backtrace to the specified print stream.
void printStackTrace(PrintWriter s)
Prints this throwable and its backtrace to the specified print writer.
void setStackTrace(StackTraceElement[] stackTrace)
Sets the stack trace elements that will be returned by getStackTrace() and printed by printStackTrace() and related methods.
String toString()
Returns a short description of this throwable.

Runtime exceptions JVM

- OutOfMemoryError - во время создания объекта (объекта – класса, массива, временного объекта при конкатенации строк)
- NegativeArraySizeException - во время создания объекта – массива.
- NullPointerException - во время использования (доступ к полю или вызов метода) ссылки или выражения результатом которого является ссылочный тип, имеющего значение null
- NullPointerException - при обращении к элементу массива в случае если ссылка на массив содержит null
- ArrayIndexOutOfBoundsException - в случае выхода за границы массива
- ClassCastException - при попытке приведения несовместимых типов.
- ArithmeticException - деление на ноль при выполнении операторов «/», «%»
- ArrayStoreException - при попытке присваивания элементу массива значения несовместимого ссылочного типа.

Try with resources. Step №1

```
OutputStream stream = openOutputStream();
// do something with stream
stream.close();
```

Try with resources. Step №2

```
OutputStream stream = openOutputStream();
try {
   // do something with stream
} finally {
   stream.close();
}
```

Try with resources. Step №3

```
OutputStream stream = openOutputStream();
try {
   // do something with stream
} finally {
   try {
      stream.close();
   } catch (Throwable unused) {
      // ignore
   }
}
```

Try with resources. Step №4

```
OutputStream stream = openOutputStream();
Throwable mainThrowable = null;
try {
   // do something with stream
} catch (Throwable t) {
   // save exception
   mainThrowable = t;
   // and throw new exception
   throw t;
} finally {
    if (mainThrowable == null) {
       // All correct. Call only close()
       stream.close();
    }
    else {
       try {
         stream.close();
       } catch (Throwable unused) {
          // } ignore or logging

    }
}
```

Try with resources. Step №5

```
try (OutputStream stream = openOutputStream()) {
    // do something with stream
}
```

```
Exception in thread "main" java.lang.RuntimeException: Main exception
        at A$1.write(A.java:16)
        at A.doSomething(A.java:27)
        at A.main(A.java:8)
        Suppressed: java.lang.RuntimeException: Exception on close()
                at A$1.close(A.java:21)
                at A.main(A.java:9)
```

Try with custom resource

```
class OpenDoor implements AutoCloseable {
        public OpenDoor() throws Exception {
                System.out.println("The door is open.");
        }
        public void swing() throws Exception {
                System.out.println("The door is becoming unhinged.");
                throw new SwingException();
        }

        public void close() throws Exception {
                System.out.println("The door is closed.");
                throw new CloseException(); /* throwing CloseException */
        }
}
```

Try with custom resource

```
public class TryWithResources {
        public static void main(String[] args) throws Exception {

                try ( OpenDoor door = new OpenDoor() ) {
                        door.swing(); /*this throws a SwingExecption*/
                }
                catch (Exception e) {
                        System.out.println("Is there a draft? " + e.getClass());
                        int suppressedCount = e.getSuppressed().length;
                        for (int i=0; i<suppressedCount; i++){
                                System.out.println("Suppressed: " + e.getSuppressed()[i]);
                        }
                }
                finally {
                        System.out.println("I'm putting a sweater on, regardless. ");
                }
        }
}
```

```
The door is open.
The door is becoming unhinged.
The door is closed.
Is there a draft? class SwingException
Suppressed: CloseException
I'm putting a sweater on, regardless.
```