Java Training

Day 8: Errors, Exceptions and Exception Handling

Download today’s slides:
go/java+espresso-training/day8

Java Exceptions: this implementation of `Adder` is missing `args` checking

```
public class Adder {
     public static void main(String[] args){
          int x = Integer.valueOf(args[0]);
          int y = Integer.valueOf(args[1]);
          int sum = x + y;
          System.out.println("The answer is: " + sum);
     }
}
```

```
> java Adder 2

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 1
     at Adder.main(Adder.java:4)
```

Previous implementations used array length checking and conditional logic

```
if (args.length > 0) { ...
```

to avoid throwing `ArrayIndexOutOfBoundsException`, but you can *catch* exceptions at runtime *after* they have occurred

Try/Catch blocks

```
public class ArgsExcept {
    public static void main(String[] args){
        try {
            int x = Integer.valueOf(args[0]);
            int y = Integer.valueOf(args[1]);
            int sum = x + y;
            System.out.println("The answer is: " + sum);

        } catch (ArrayIndexOutOfBoundsException ex) {
            System.out.println("Error: too few arguments");
        }
    }
}
```

Exceptions thrown within the `try` block are passed to the appropriate `catch` block

```
> java Adder 2

Error: too few arguments
```

Try/Catch structures

```
try {
        ...
    } catch (ExceptionType1 e) {
        ...
    } catch (ExceptionType2 e) {
        ...
    }
```

To handle multiple exception types you can use multiple catch blocks or handle multiple types using a single blocks using the '|' operator (or use a more general exception class)

```
try {
        ...
    } catch (ExceptionType1 | ExceptionType 2 e) {
        ...
    }
```

Coding Exercise #1

Non-numeric arguments(e.g. “xx”) will generate a `NumberFormatException`
Modify the `try/catch` blocks in `Adder` so this exception type is also handled

```
public class Adder {
    public static void main(String[] args){
        try {
            int x = Integer.valueOf(args[0]);
            int y = Integer.valueOf(args[1]);
            int sum = x + y;
            System.out.println("The answer is: " + sum);

        } catch (ArrayIndexOutOfBoundsException ex) {
            System.out.println("Error: too few arguments");
        }
    }
}
```

Possible Solution

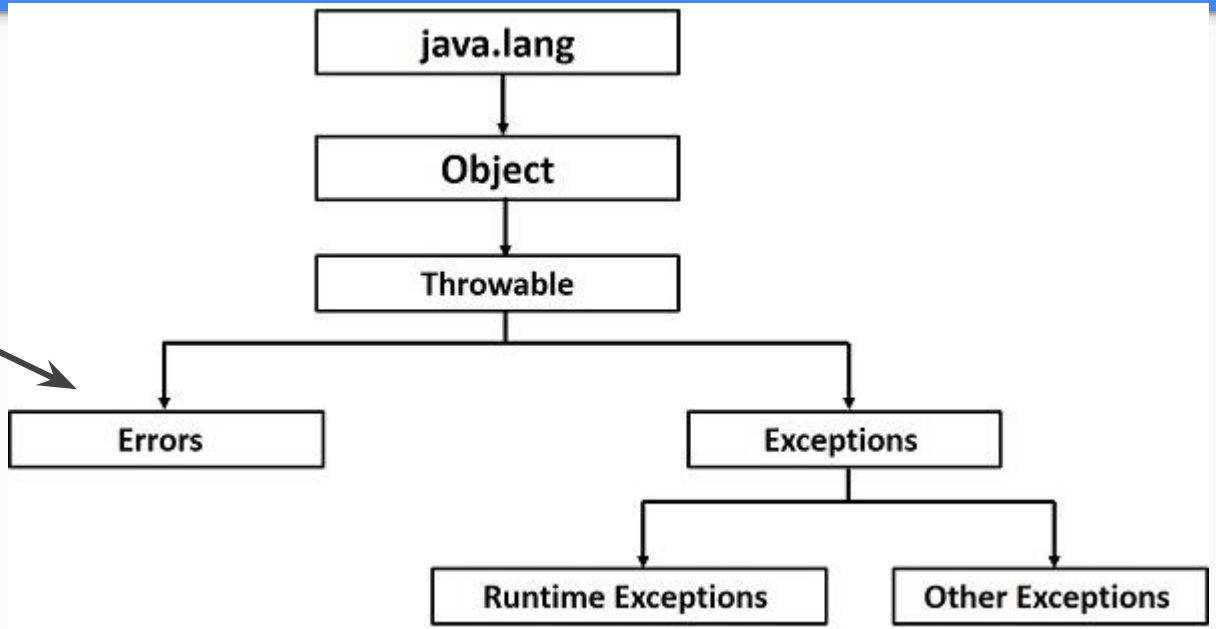
```
public class ArgsExcept {
        public static void main(String[] args){
                try {
                        int x = Integer.valueOf(args[0]);
                        int y = Integer.valueOf(args[1]);
                        int sum = x + y;
                        System.out.println("The answer is: " + sum);
                } catch (ArrayIndexOutOfBoundsException | NumberFormatException ex){
                        System.out.println("Error: bad input " + ex);
                }
        }
}
```

Java **Errors & Exceptions**

Errors *cannot* be caught

These are usually environmental (JVM state)

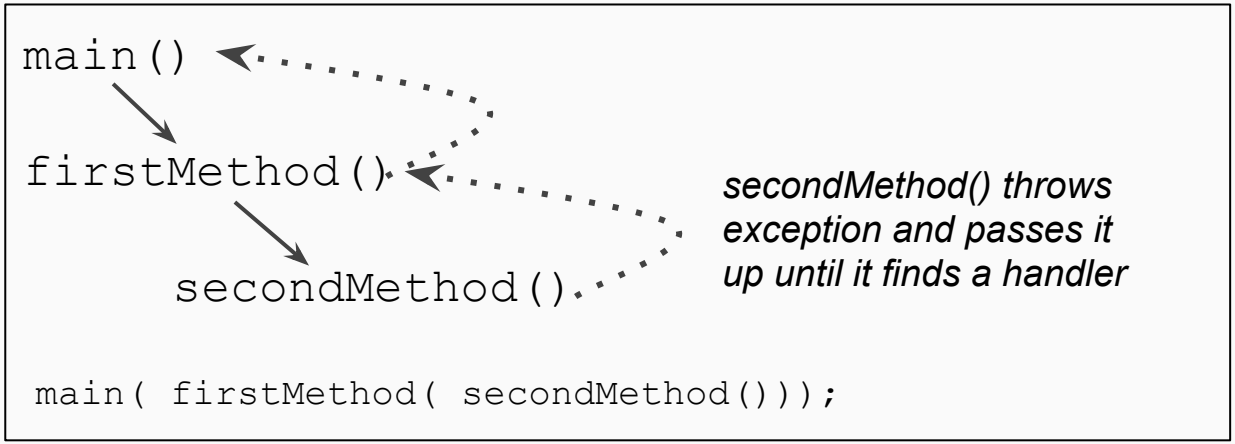
`OutOfMemoryError`
`StackOverflowError`



`ArrayIndexOutOfBoundsException`
and
`NumberFormatException`

Java Exceptions

Exceptions are *passed* up the **call-stack**



`NumberFormatException` is passed up to `main()` when `Adder` fails

```
> java Adder xx yy
Exception in thread "main" java.lang.NumberFormatException: For input string: "xx"
     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.valueOf(Integer.java:983)
     at Adder.main(Adder.java:3)
```

Coding Exercise #2

Add a `try/catch` block to `main()` to handle any `ArrayIndexOutOfBoundsException` generated by `adder()`

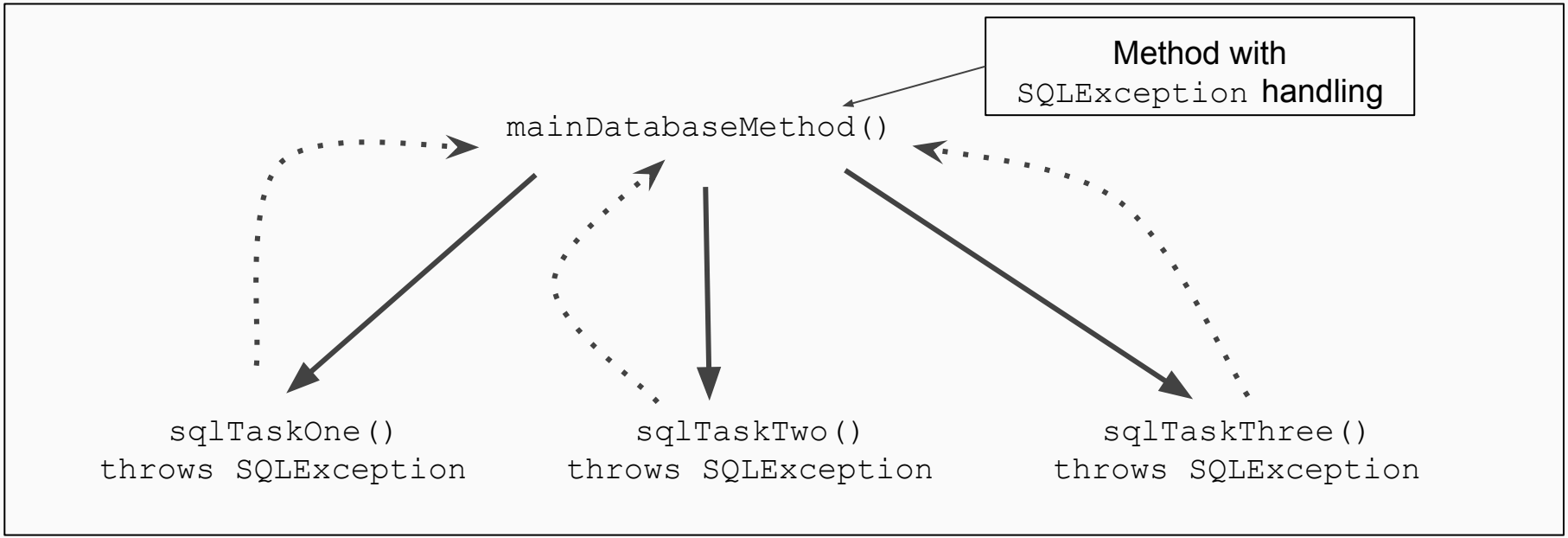
```
public class PassedExcept {
        public static void main(String[] args){
                adder(args);
        }
        static void adder(String[] args){
                int x = Integer.valueOf(args[0]);
                int y = Integer.valueOf(args[1]);
                int sum = x + y;
                System.out.println("The answer is: " + sum);
        }
}
```

Possible Solution

```
public class PassedExcept {
        public static void main(String[] args){
                try {
                        adder(args);
                } catch (ArrayIndexOutOfBoundsException ex) {
                        System.out.println("Needs more args! " + ex);
                }
        }
        static void adder(String[] args){
                int x = Integer.valueOf(args[0]);
                int y = Integer.valueOf(args[1]);
                int sum = x + y;
                System.out.println("The answer is: " + sum);
        }
}
```

More, Java **Errors** & **Exceptions**

Exception passing allows handling to be centralized



`mainDatabaseMethod()` wraps the calls to `sqlTaskOne()`, `sqlTaskTwo()` and `sqlTaskThree()` in a `try` block and catch `SQLException` (see next slide)

Checked vs **Unchecked** Exceptions

If a method may throw a **checked exception** (like `SQLException`), it must be declared:

```
public void mainDatabaseMethod() {
    try {
        sqlTaskOne();
        sqlTaskTwo();
        sqlTaskThree();
    } catch (SQLException ex) {
        ...
    }
}
void sqlTaskOne() throws SQLException {...}
void sqlTaskTwo() throws SQLException {...}
void sqlTaskThree() throws SQLException {...}
```

Runtime exceptions, like `ArrayIndexOutOfBoundsException` and `NumberFormatException` are unchecked exceptions, so they don't require `throw` declarations (but they are still passed up the call-stack when they occur)

DIY Checked Exceptions: create your own

```
public class DIYexcept {
        public static void main(String[] args){
                try {
                        testMethod();
                } catch (Exception ex) {
                        System.out.println("got it! " + ex);
                }
        }
        static void testMethod() throws MyException {
                throw new MyException("very bad thing");
        }
}

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

This is a checked exception so this won't compile without `throws` declaration

```
> javac DIYexcept.java
> java DIYexcept
got it! MyException: very bad thing
```

Subclasses of `Exception` other than `RuntimeException` are *checked* by the compiler