Java Workbook

07. Exceptions

~by Andrzej "Andret2344" Chmiel

Exercise 1: Various cases of `IOException`

Throw as many exceptions being subclasses of the `IOException` as possible, not importing any of them nor using qualified access. In the main method, use `throws Exception`.

Exercise 2: Various strange cases of `IOException`

Throw as many exceptions being subclasses of the `IOException` as possible, not importing any of them nor using qualified access, but using the `throw` keyword. In the main method, use `throws Exception`.

Exercise 3: What the exception, dude?

Given the following code:

```
class MainWhatTheExceptionDude {
    public static void main(final String[] args) {
        try {
            safeThrow(5);
        } catch (final Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    private static void safeThrow(final int value) {
        throwException(value + 1);
    }

    private static void throwException(final int value) throws IOException {
        throw new IOException("The value: " + value * 2);
    }
}
```

Change only the body of the `safeThrow` method so that the code compiles and prints out this:

```
The value: 12
```

Exercise 4: Things that do matter

Given the following code:

```
class MainThingsThatDoMatter {
    public static void main(final String[] args) {
        try {
            catchBoth(true);
        } catch (final IOException e) {
            System.out.println("Exception in main: " + e.getClass().getName());
        }
        try {
            catchBoth(false);
        } catch (final IOException e) {
            System.out.println("Exception in main: " + e.getClass().getName());
        }
    }

    private static void catchBoth(final boolean value) throws IOException {
        try {
            throwAnException(value);
        } catch (final IOException e) {
            System.out.println("IOException caught");
        } catch (final FileNotFoundException e) {
            System.out.println("FileNotFoundException caught");
        }
    }

    private static void throwAnException(final boolean value) throws IOException {
        if (value) {
            throw new FileNotFoundException();
        }
        throw new IOException();
    }
}
```

Change only the body of the `catchBoth` method so that the code compiles and prints out this:

```
FileNotFoundException caught
Exception in main: java.io.FileNotFoundException
IOException caught
Exception in main: java.io.IOException
```

Exercise 5: Missing exceptions

Given the following code:

```
class A {
    void foo() throws Ex1 {
        throw new Ex1("Foo in class A");
    }

    void bar() {
        throw new Ex2("Bar in class A");
    }
}

class B extends A {
    @Override
    void foo() throws Ex3 {
        throw new Ex3("Foo in class B");
    }

    @Override
    void bar() {
        throw new Ex4("Bar in class B");
    }
}

class MainMissingExceptions {
    public static void main(final String[] args) {
        final B b = new B();
        try {
            b.foo();
        } catch (Ex1 ex) {
            System.out.println(ex.getMessage());
        }

        try {
            b.bar();
        } catch (Ex2 ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

Create missing exception classes so the code compiles and prints out this:

```
Foo in class B
Bar in class B
```

Exercise 6: Hello and Goodbye

Given the following code:

```
class MainHelloGoodbye {
    public static void main(final String[] args) throws IOException {
        System.out.println("hello");
        throw new FileNotFoundException();
        System.out.println("goodbye");
    }
}
```

Not declaring any variable and not changing lines order fix the code, so it compiles and prints out this:

```
hello
goodbye
Exception in thread "main" java.io.FileNotFoundException
```

Lines can be in slightly different order.

Exercise 7: What the exception, dude?

Given the following code:

```
class MainWhatTheExceptionDude {
    public static void main(final String[] args) {
        try {
            safeThrow(5);
        } catch (final Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    private static void safeThrow(final int value) {
        throwException(value + 1);
    }

    private static void throwException(final int value) throws IOException {
        throw new IOException("The value: " + value * 2);
    }
}
```

Change only the body of the `safeThrow` method so that the code compiles and prints out this:

```
The value: 12
```

Exercise 8: Pattern matching

Given the following code:

```
record EmailAddress(String value) {
    EmailAddress {
        if (!MainPatternMatching.EMAIL_REGEX.matcher(value).matches()) {
            throw new IllegalArgumentException(
                    String.format("Email doesn't match the regex \"%s\"",
                            MainPatternMatching.EMAIL_REGEX.pattern()));
        }
    }
}

class MainPatternMatching {
    public static final Pattern EMAIL_REGEX =
            Pattern.compile("^\\S+@\\S+\\.\\S{2,}$");

    public static void main(final String[] args) {
        try {
            System.out.println(new EmailAddress("abc"));
        } catch (final Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

Fix the `main` method code, so it doesn't use try-catch and the outcome is the same.

Exercise 9: Optional exceptions

Given the following code:

```
class MainOptionalExceptions {
    public static void main(final String[] args) {
        System.out.println(mapIt("", 0));
        System.out.println(mapIt("a", 0));
        System.out.println(mapIt("a", 1));
    }

    private static String mapIt(final String text, final int value) {
        return Optional.of(text)
                .filter(s -> filter(s, value))
                .orElse(onlyZero(value));
    }

    private static boolean filter(final String s, final int value) {
        return s.length() == value;
    }

    private static String onlyZero(final int value) {
        if (value == 0) {
            return "zero";
        }
        throw new IllegalArgumentException("Not zero!");
    }
}
```

Changing only the body of the `mapIt` method but sticking to the `Optional` chain, fix the code, so it works correctly and doesn't throw any exceptions.

Exercise 10: Failed creation

Given the following code: Create a `Person` entity that contains `firstName`, `lastName`, and `birthDate` that matches the following criteria:

- The first name must start with an uppercase letter and all the following letters must be lowercased.
- The last name must contain only uppercased and lowercased letters and a dash.
- The birthdate cannot be in the future.
- The object can't be created if anything is wrong.