Java training

Exception handling, debugging an app

Session overview

- **Exception handling**
  - General overview
  - Exception types - checked and unchecked
  - `try / catch / finally`
  - Multiple catch blocks
  - `try with resources`
- **Debugging an app**

Exception

- **Exception** - event that disrupts the normal flow of instructions
    - Examples:
        - Entering invalid data
        - Network, file or database errors
        - Programming bugs
- **Call stack** - the stack of methods through which a program is executed
    - Usually begins with the ‘`main`’ method
    - Ends with the method and line where the exception occurred

Examples

- **NullPointerException**:

```
    private String name = null;          // a not initialized variable
    System.out.println(name.length()); // → NullPointerException
```

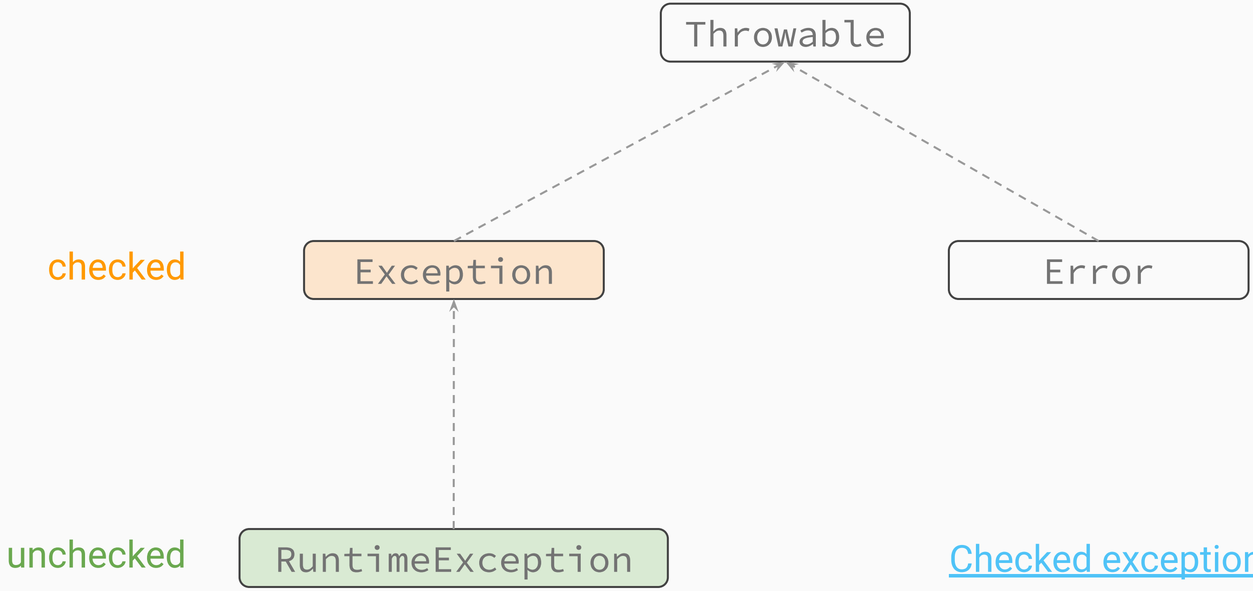
- **NumberFormatException**

```
    private String id = “23x”;
    private int parsed = Integer.parseInt(id);
                                        // → NumberFormatException
```

Exception types

- **Checked** (compile-time) exceptions - occurs at the compile time
  - Need to be caught or re-thrown
- **Unchecked** (runtime) exceptions - occur when the program is executing
  - Don’t need to be caught
- **Errors** - problems that arise beyond the control of the programmer
  - Indicates *serious problems* that a reasonable application should not try to catch
  - A serious abnormal condition in the program

Exceptions hierarchy



[Checked exceptions - Java's biggest mistake]

Exception handling

- Handled via **`try / catch`** blocks
- Example:

```
try {
    int value = Integer.parseInt("23x");
                                        // statement(s) which can throw exception(s)
} catch (Exception ex) {
    // handle exception
}
```

- Can be further re-thrown (if needed)

Exception handling

- `try / catch` blocks
  - Used to handle most exceptions
    - Mandatory for checked exceptions
    - Recommended for unchecked exceptions
  - Can handle multiple exception types

    ```
    catch (AnException | AnotherException ex)
    ```
- `finally` block
  - Executed regardless if the exception occurs or not
  - Not mandatory

Multiple catch blocks

- Some exception types need to be handled differently
- Multiple catch blocks can be used in a `try / catch` block
- Example:

```
try {
        // processing which may cause exceptions
} catch (NullPointerException ex) {
        // processing the null pointer
} catch (Exception ex) {
        // processing the generic exception
}
```

Complete example

```
int value = 0;
try {
    value = Integer.parseInt(“23”);
} catch (Exception ex) {
    value = -1;
    ex.printStackTrace();  // prints the entire call stack
} finally {
    System.out.println(“The value is “ + value);
}
```

Using checked exceptions

- Methods can be declared as **throwing** (checked) exceptions

```
void processProduct(Product product) throws Exception;
```

- The calling method must either:
  - **Catch the exception(s)** - using a `try / catch` block
  - **Re-throw the exception(s)** - propagate or change the exception type

```
throw new Exception(ex.getMessage());
```

- **Advice** - catch and process it **as soon as possible** (processing wise)

Hands-on

- Using checked exceptions
  - Throwing
  - Catching them in `try / catch` blocks
  - Changing the exception type

Using unchecked exceptions

- Not mandatory to be caught; advisable to use a handler `(try-catch)` for them
- Thrown once, will ‘traverse’ the call stack until a handler will be met
- If no handler will be met - the program execution *will be interrupted*

Exceptions - usage advice

- Use **unchecked** exceptions, as much as possible
    - **Throw** them from the method where the exceptions occurred
    - **Catch** and **handle** them from a single place - centralized exception handling
    - **Define** and **use different exception types**, based on the business logic
- For **checked** exceptions
    - Do ***NOT*** perform logic in the `catch` and `finally` blocks
    - ‘**`catch`**’ *should* be used just for logging the errors and sending error reports
    - ‘**`finally`**’ *should* be used just for closing resources (further discussed)

try with resources

- Since Java 7, automatic resource closing / releasing can be used → ‘try with resources’ blocks

```
try (FileReader fr = new FileReader(file);
    BufferedReader br = new BufferedReader(fr)) {
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
} catch (IOException ex) {
    ex.printStackTrace();
}
```

Both streams will be closed automatically

Debugging an application

- **Debugging** - investigating and fixing a bug
- Mostly done from the IDE
- Main modes:
  - Adding logging statements (messages)
    - + Can be watched and changed dynamically
    - - Read-only access
    - - May be difficult to follow
  - Running the app in debug mode
    - + Dynamic tracing of the program execution
    - - Requires debugging mode access (local / remote)

Debugging from the IDE

- Breakpoints - execution points where the processing will be paused
- Watches - variables watched for changes

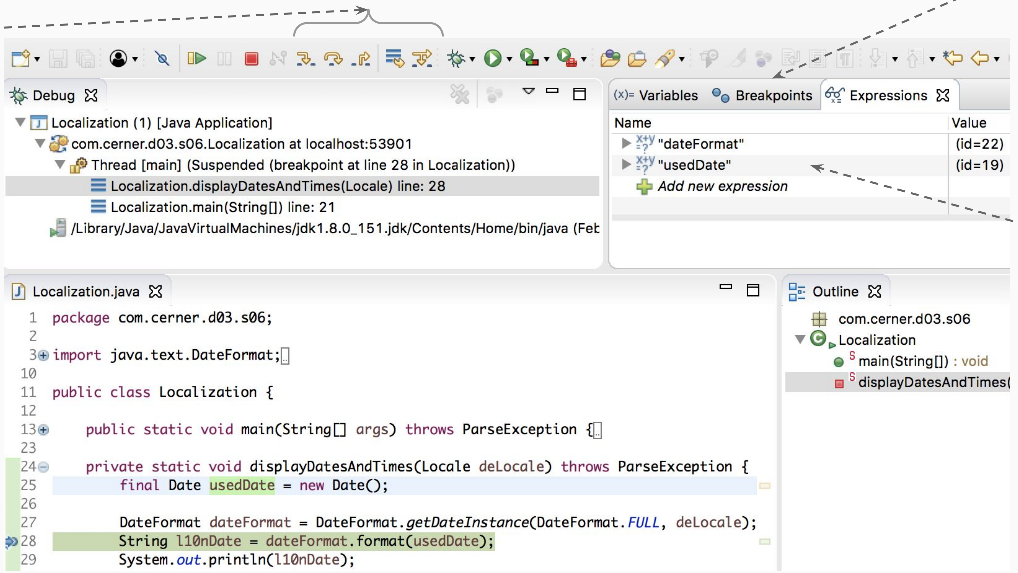
Execution continuation mode

Thread selection section

Breakpoint

Breakpoints view

Watches



```
Localization (1) [Java Application]
  com.cerner.d03.s06.Localization at localhost:53901
    Thread [main] (Suspended (breakpoint at line 28 in Localization))
      Localization.displayDatesAndTimes(Locale) line: 28
      Localization.main(String[]) line: 21
  /Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home/bin/java (Feb
```

Name	Value
"dateFormat"	(id=22)
"usedDate"	(id=19)
Add new expression	

```
package com.cerner.d03.s06;

import java.text.DateFormat;

public class Localization {

    public static void main(String[] args) throws ParseException {

    private static void displayDatesAndTimes(Locale deLocale) throws ParseException {
        final Date usedDate = new Date();

        DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.FULL, deLocale);
        String l10nDate = dateFormat.format(usedDate);
        System.out.println(l10nDate);
```

```
com.cerner.d03.s06
  Localization
    main(String[]) : void
    displayDatesAndTimes(
```

Hands-on →

Hands-on

- Use various `try / catch / finally` statements
    - Catching null assignments
    - Parsing various values
- Use checked and unchecked exceptions
- Define and throw your own exception types:
    - Create a `ProductNotFound` exception, use it from a `ProductService`

Our use-case - a `ProductService`

- The `ProductService` contains CRUD operations for Products
- Exception throwing use-cases:
  - **C**reating a product:
    - Invalid name or price
  - **R**eading the products:
    - No products are available
    - There is no product with a given ID
  - **U**pdating a product:
    - There is no product with the given ID
    - Invalid name or price
  - **D**eleting a product:
    - There is no product with the given ID

Further reading

- https://advancedweb.hu/2018/02/06/debug/
- http://www.vogella.com/tutorials/EclipseDebugging/article.html
- https://www.youtube.com/watch?v=9gAjIQc4bPU

+ countless other articles

Q&A session

1. You ask, I answer
2. I ask, you answer