Modul 7

Fil- og unntaks-håndtering

Uke7Xtra

Nokke helt annet..

Bruk av AI i eksamen:
https://kristiania.instructure.com/courses/13605/files/1535361?module_item_id=525224

Endring i oppsett ekstra time : tid og format (mentimeter).

Filhåndtering

Lese fra fil

```
File file = new File("eksempel.txt");
Scanner input = new Scanner(file);
while(input.hasNextLine()){
        String s = input.nextLine();
        System.out.println(s);
}
input.close();
```

Filhåndtering

Lese fra fil

```
File file = new File("eksempel.txt");
Scanner input = new Scanner(file);
```

Prøv selv … Code along

Filhåndtering

Lese fra fil

```
File file = new File("eksempel.txt");
Scanner input = new Scanner(file);
```

Prøv selv … Code along

Fil – og unntakshåndtering

- vi må lære dette samtidig fordi..

```
Scanner scanFile = new Scanner(file);
//while (scanFile.hasNextLi
//    System.out.println(sc

//}
}
}
```

Unhandled exception: java.io.FileNotFoundException

Add exception to method signature ⌥⇧⏎ More actions... ⌥⏎

```
© java.util.Scanner
public Scanner(
    @NotNull ↗ java.io.File source
)
throws java.io.FileNotFoundException
```

Constructs a new `Scanner` that produces values scanned from the specified file. Bytes from the file are converted into characters using the underlying platform's default charset.

Params: `source` – A file to be scanned

```
Scanner scanFile = new Scanner(file);
```

- Add exception to method signature
- Surround with try/catch
- Split into declaration and assignment
- Surround with try-with-resources block

Press F1 to toggle preview

```
4  import java.io.FileNotFoundException;
10 try {
11     Scanner scanFile = new Scanner(file);
12 } catch (FileNotFoundException e) {
13     throw new RuntimeException(e);
14 }
```

```
9   System.in.close();
10  Scanner sca
11
12  }
13 }
14
```

Unhandled exception: java.io.IOException

Add exception to method signature ⌥⇧⏎ More actions... ⌥⏎

```
© java.io.InputStream
@Contract(mutates = "this,io") ↗
public void close()
throws java.io.IOException
```

Closes this input stream and releases any system resources associated with the stream.

The `close` method of `InputStream` does nothing.

Specified by: `close` in interface `Closeable`

Throws: `IOException` – if an I/O error occurs.

< corretto-17 >

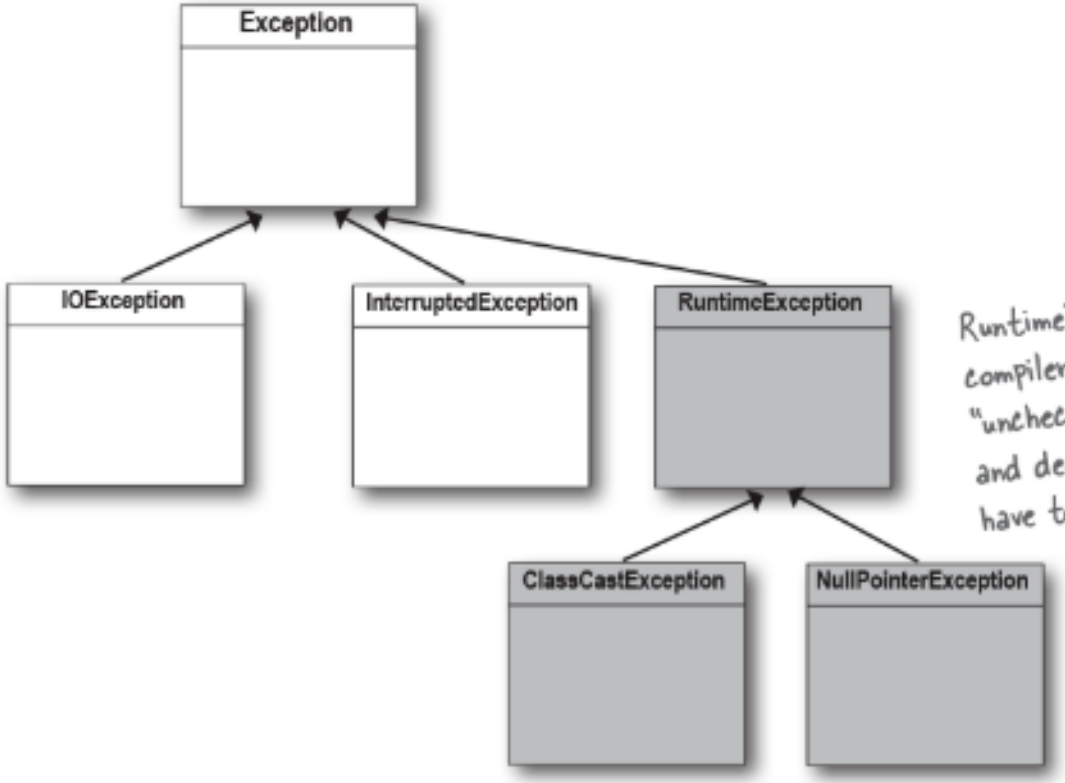
Riktig eller feil?

- ⊘ En try-blokk MÅ alltid følges av en catch-blokk OG an finally-blokk.
- ⊘ Kun «kompilator sjekket» unntak kan fanges opp
- ✓ En enestående try-blokk kan ha mange forskjellige catch blokker
- ⊘ Når du skriver en metode som kunne forårsake et sjekket unntak MÅ du pakke inn den risikable koden i en try/catch- blokk

Checked vs unchecked exceptions

The compiler checks for everything except RuntimeExceptions.

Exceptions that are NOT subclasses of RuntimeException are checked for by the compiler. They're called "checked exceptions."



RuntimeExceptions are NOT checked by the compiler. They're known as (big surprise here) "unchecked exceptions." You can throw, catch, and declare RuntimeExceptions, but you don't have to, and the compiler won't check.

Checked vs unchecked exceptions

The compiler guarantees:

- ① If you *throw* an exception in your code, you *must* declare it using the *throws* keyword in your method declaration.
- ② If you *call* a method that throws an exception (in other words, a method that *declares* it throws an exception), you must *acknowledge* that you're aware of the exception possibility. One way to satisfy the compiler is to wrap the call in a try/catch. (There's a second way we'll look at a little later in this chapter.)

Checked vs unchecked exceptions

- Hands-on: try– catch -finally

```
try {
  turnOvenOn();
  x.bake();
} catch (BakingException e) {
  e.printStackTrace();
} finally {
  turnOvenOff();
}
```

Ingen garbage-collection for ressurser:

Classes which utilize non-memory resources should provide ways to explicitly allocate/deallocate those resources. We need to explictly call `close()` methods for deallocation of file descriptors in `finally{}`, as it will execute whether or not an exception is thrown.

- **try-with-resources**

Unntak

The `try`-with-resources statement is a `try` statement that declares one or more resources. A *resource* is an object that must be closed after the program is finished with it. The `try`-with-resources statement ensures that each resource is closed at the end of the statement. Any object that implements `java.lang.AutoCloseable`, which includes all objects which implement `java.io.Closeable`, can be used as a resource.

https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Unntak

- throw

```
public class Laundry {
  public void doLaundry() throws PantsException, LingerieException {
    // code that could throw either exception
  }
}
```

This method declares two, count 'em, TWO exceptions.

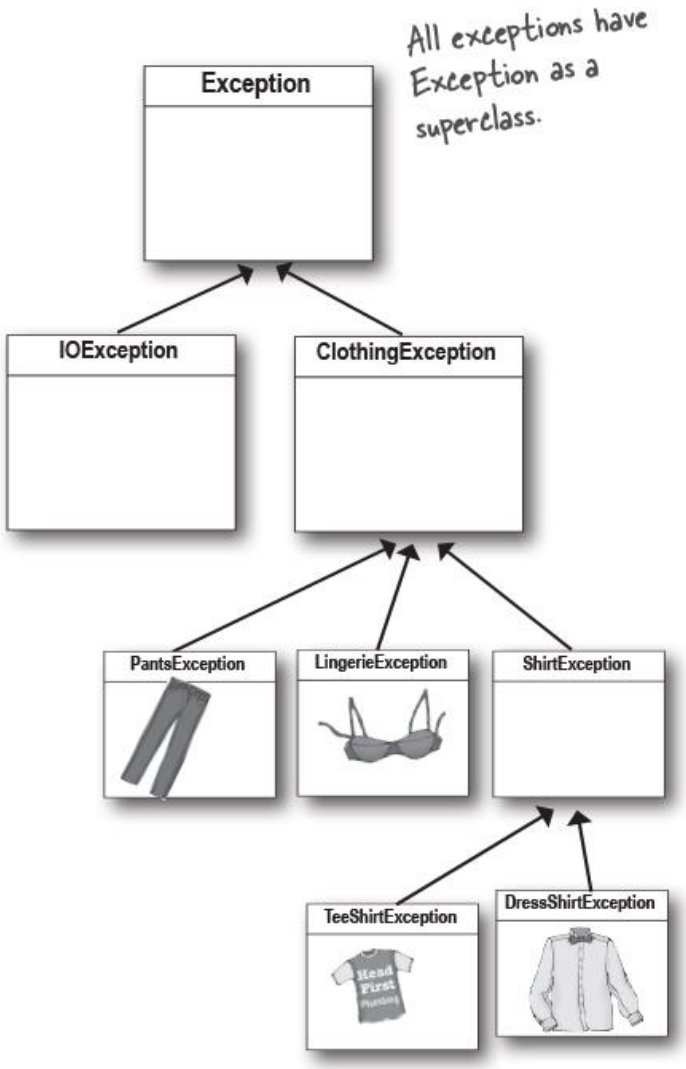
```
public class WashingMachine {
  public void go() {
    Laundry laundry = new Laundry();
    try {
      laundry.doLaundry();
    } catch (PantsException pex) {
      // recovery code

    } catch (LingerieException lex) {
      // recovery code
    }
  }
}
```

If doLaundry() throws a PantsException, it lands in the PantsException catch block.

If doLaundry() throws a LingerieException, it lands in the LingerieException catch block.

Unntak

- throw



ARV !
SUPERKLASSE !!
POLYMORFI !!!

Unntak

- throw

```
public void doLaundry() throws ClothingException {
```

Declaring a ClothingException lets you throw any subclass of ClothingException. That means doLaundry() can throw a PantsException, LingerieException, TeeShirtException, and DressShirtException without explicitly declaring them individually.

ARV !
SUPERKLASSE !!
POLYMORFI !!!

- ② **You can CATCH exceptions using a superclass of the exception thrown.**

```
try {
  laundry.doLaundry();
} catch(ClothingException cex) {
  // recovery code
}
```

Can catch any ClothingException subclass

```
try {
  laundry.doLaundry();
} catch(ShirtException shex) {
  // recovery code
}
```

Can catch only TeeShirtException and DressShirtException

Unntak

Unntak - oppsummert

- Når vi benytter (kaller på) metoder som kaster checked exceptions, må vi bestemme om vi vil kaste det mulige unntaket videre, eller behandle det
  - Behandle det (try/catch/finally)
  - Kaste det videre: presisere i metoden at den kaster unntak videre med bruk av *throws*
    - Eks:

```
public void method
        throws IOException{}
```