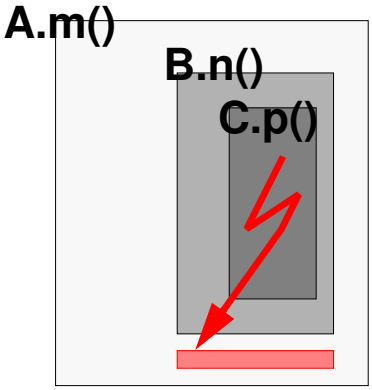
23 Error Handling

- Exit program (`System.exit()`)
  - ◆ usually a bad idea
- Output an error message
  - ◆ does not help to recover from the error
- Special error return
  - ◆ Constructors do not have a return value
  - ◆ What if method uses the full range of the return type?
- Call a user defined error handler
  - ◆ awkward
  - ◆ What must this method do?
- Exceptions!

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.1 What is Exception Handling?

- Transfer control from error origin to error handler



- Responsibilities:
  - ◆ Code author can detect the error but doesn't know how to handle it.
  - ◆ Code user can handle the error but cannot detect it.

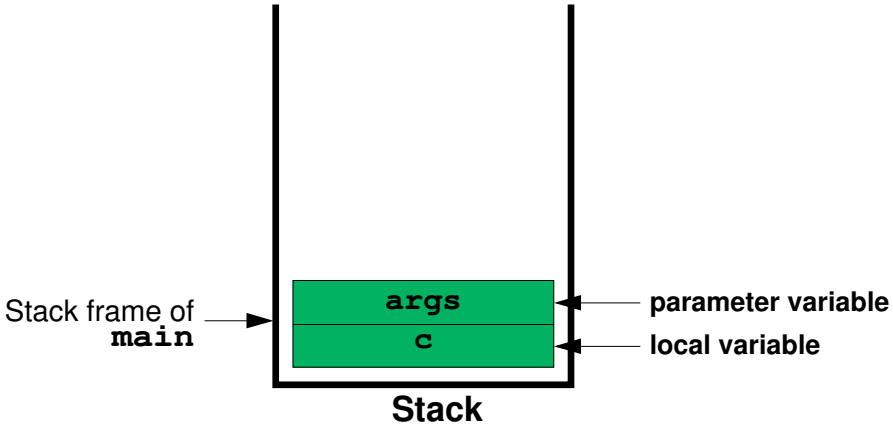
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.1 What happens when a method is called?

```
class Customer {
  void createAccount(Bank bank) {
    Account account = new Account();
    bank.newAccount(account, 5);
  }
}
```

```
class Bank {
  void newAccount(Account a, int i) {
        int counter = 0;
    ...
  }
}
```

```
class Main {
  public static void main(String args[]) {
        Customer c = new Customer();
        c.createAccount(new Bank());
  }
}
```



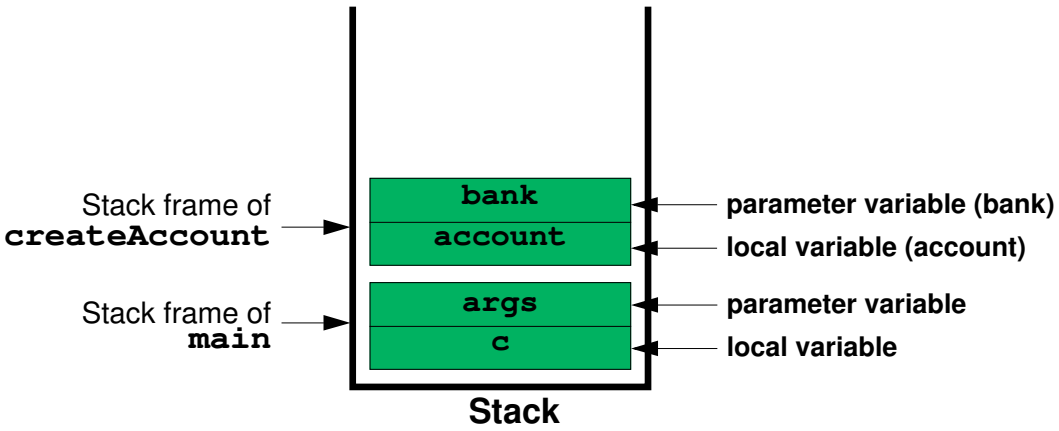
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.1 What happens when a method is called?

```
class Customer {
  void createAccount(Bank bank) {
    Account account = new Account();
    bank.newAccount(account, 5);
  }
}
```

```
class Bank {
  void newAccount(Account a, int i) {
        int counter = 0;
    ...
  }
}
```

```
class Main {
  public static void main(String args[]) {
        Customer c = new Customer();
        c.createAccount(new Bank());
  }
}
```



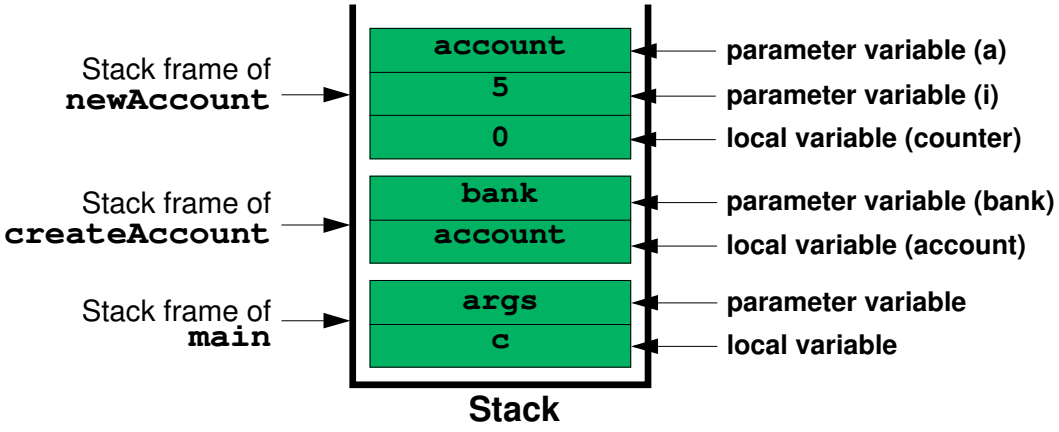
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.1 What happens when a method is called?

```
class Customer {
  void createAccount(Bank bank) {
    Account account = new Account();
    bank.newAccount(account, 5);
  }
}
```

```
class Bank {
  void newAccount(Account a, int i) {
        int counter = 0;
    ...
  }
}
```

```
class Main {
  public static void main(String args[]) {
        Customer c = new Customer();
        c.createAccount(new Bank());
  }
}
```



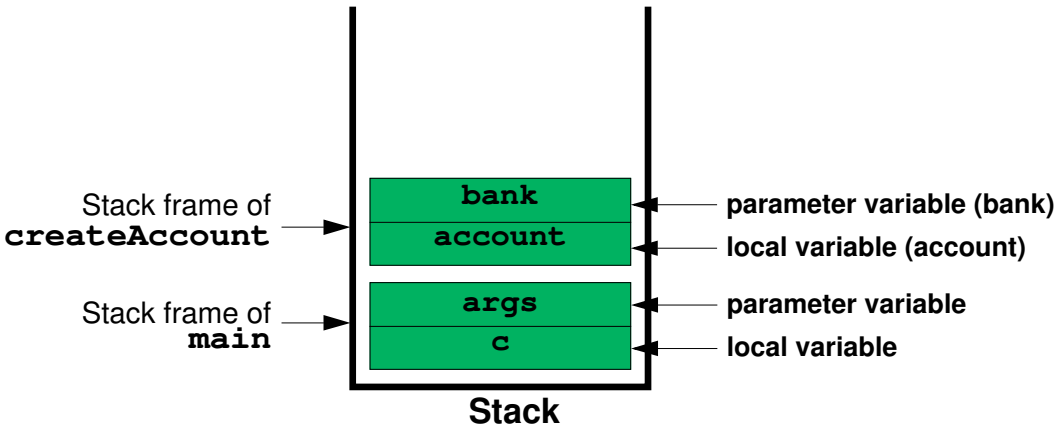
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.1 What happens when a method is called?

```
class Customer {
  void createAccount(Bank bank) {
    Account account = new Account();
    bank.newAccount(account, 5);
  }
}
```

```
class Bank {
  void newAccount(Account a, int i) {
        int counter = 0;
    ...
  }
}
```

```
class Main {
  public static void main(String args[]) {
        Customer c = new Customer();
        c.createAccount(new Bank());
  }
}
```



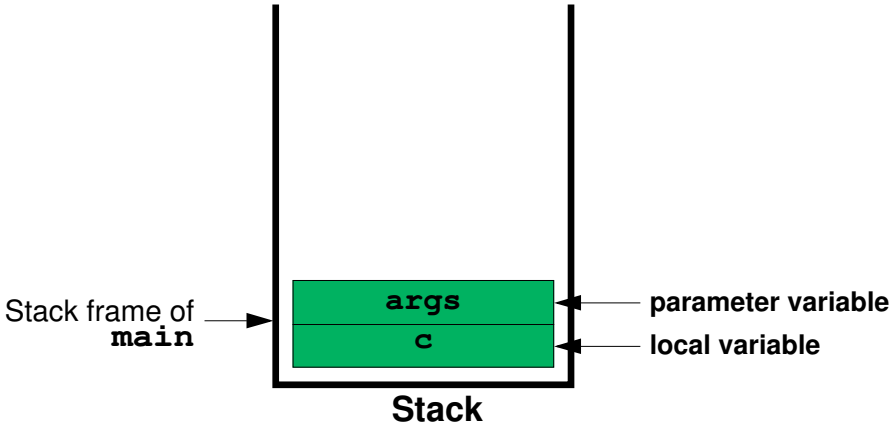
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.1 What happens when a method is called?

```
class Customer {
  void createAccount(Bank bank) {
    Account account = new Account();
    bank.newAccount(account, 5);
  }
}
```

```
class Bank {
  void newAccount(Account a, int i) {
        int counter = 0;
    ...
  }
}
```

```
class Main {
  public static void main(String args[]) {
        Customer c = new Customer();
        c.createAccount(new Bank());
  }
}
```



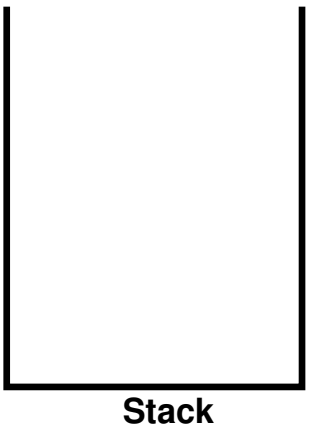
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.1 What happens when a method is called?

```
class Customer {
  void createAccount(Bank bank) {
    Account account = new Account();
    bank.newAccount(account, 5);
  }
}
```

```
class Bank {
  void newAccount(Account a, int i) {
        int counter = 0;
    ...
  }
}
```

```
class Main {
  public static void main(String args[]) {
        Customer c = new Customer();
        c.createAccount(new Bank());
  }
}
```



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.2 Try, Throw, and Catch

```
try {
    ...
    if (...) throw new MyException();
    ...
} catch(MyException e) {
    // exception handler
    ...
}
```

- use throw to thow an exception
- catch block must immediately follow try block
- there can be more than one catch block
  - ◆ catch blocks are matched in program order
- a method may not catch all exceptions
  - ◆ uncatched exceptions are automatically thrown up the stack

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.3 Finally

- the finally block is executed if the try block was entered
  - ◆ can be used to clean up in case of (un-)catched exceptions

```
try {
    ...
} catch(...) {
    // error handling
    ...
} finally {
    // release resources
    ...
}
```

- a **`finally`** block can also be used without **`catch`**

```
try {
    ...
    if (...) return;
    ...
} finally { ... }
```


Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.4 throws

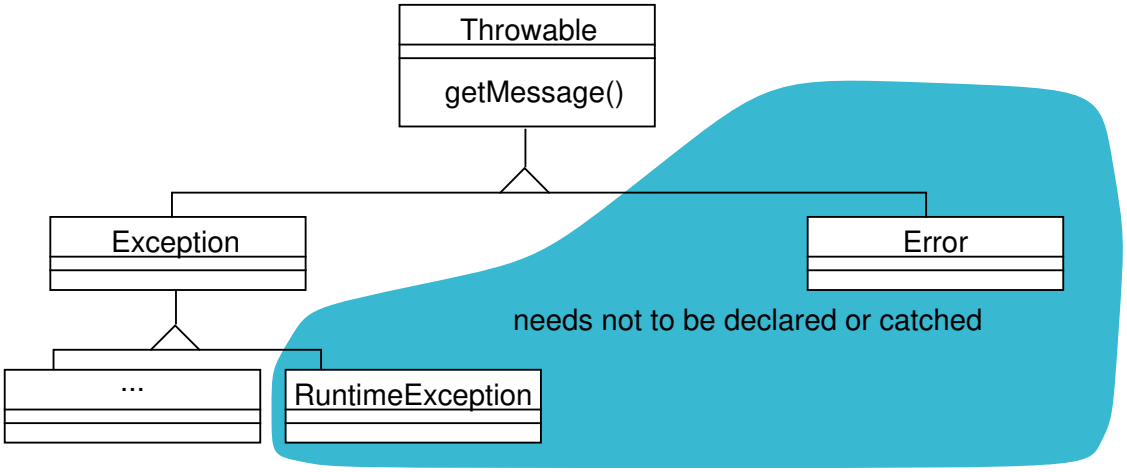
- Exceptions must be declared in method header

```
class Test {
    void m() throws MyException {
        ...
        if (...) throw new MyException();
        ...
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.5 Error Classes

- All exceptions are derived from **`Throwable`**
- Exceptions that can be expected nearly everywhere:
  - **`Error`**: Linker errors, errors in the format of class files, out of memory, ...
  - **`RuntimeException`**: array index, null pointer, illegal cast, arithmetic, ...
- Application-program exceptions are derived from **`java.lang.Exception`**



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.6 Exceptions and Inheritance: Catching

- Catching exception subclasses with multiple catch blocks
- Notice: Superclasses match all subclasses, catch superclasses at last

```
class MathException {}
class ZeroDivideException extends MathException {}
class InvalidArgException extends MathException {}
try {
    ...
} catch(ZeroDivideException e) {
    ...
} catch(InvalidArgException e) {
    ...
} catch(MathException e) {
    ...
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.6 Example

```
class TestException extends Exception {
  public TestException(String s) {super(s);}
}


public class Test {

  public void hello() throws TestException {
    if (...) throw new TestException("...an error description...");
  }

  public void testIt() {
    try {
      hello();
      ...
    }
    catch (TestException t) {
      System.out.println("Exception raised:" + t.getMessage());
    }
    finally {
      // clean up
    }
  }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.7 Exceptions and Inheritance: Throwing

- Can overriding method throw other exceptions than the original method?
- Principle:
  - ◆ Subclasses can be used wherever a superclass is expected.
  - ◆ Subclasses are "better" than superclasses.
- This means:
  - ◆ Subclass must not throw more exceptions than superclass.
  - ◆ Subclass may throw subclasses of the superclass-thrown exceptions.
  - ◆ Subclass must not throw superclasses of the exceptions.

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.8 Example

```
class E1 extends Exception {}
class E2 extends Exception {}
class E3 extends E2 {}
class A {
    void m() throws E2 {}
}
class B extends A {
    void m() throws ??? {}
}
```

- ??? =

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.8 Example

```
class E1 extends Exception {}
class E2 extends Exception {}
class E3 extends E2 {}
class A {
    void m() throws E2 {}
}
class B extends A {
    void m() throws ??? {}
}
```

- ??? =

Wrong:	Correct:
`E1`	`E2`
`Exception`	`E3`
`...`	`none`

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

23.9 Summary

- Throwing an exception: `throw new MyException("...");`
- Block that may throw exceptions: `try { ... }`
- Handling exceptions:

```
try {
... throw new MyException("..."); ...
} catch(MyException e) { ... }
```

- Additionally: **`finally`** block
- Exceptions must be subclasses of **`Throwable`**.
- Application exceptions should be subclasses of **`Exception`**.
- Exceptions must be declared in the method header with **`throws`**

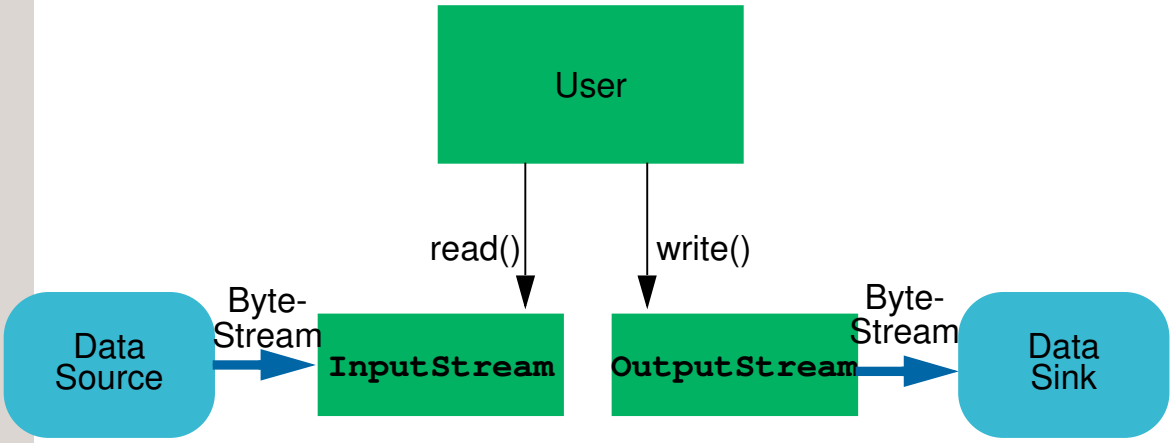
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24 The Java IO System

- Basic concept: streams
  - Byte streams (InputStream/OutputStream)
  - Charater streams (Reader/Writer)

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

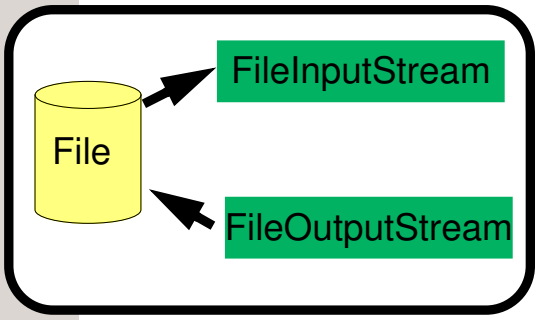
24.1 Byte Streams



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.2 Stream Specializations

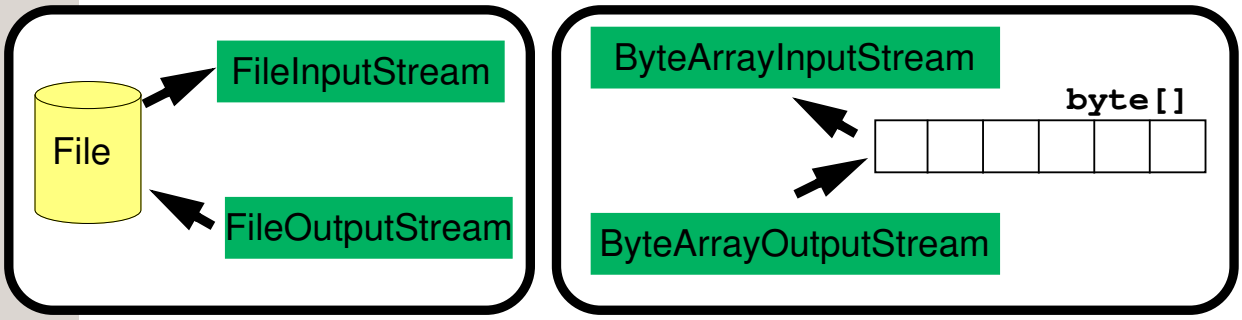
- Where do the data come from / go to?



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.2 Stream Specializations

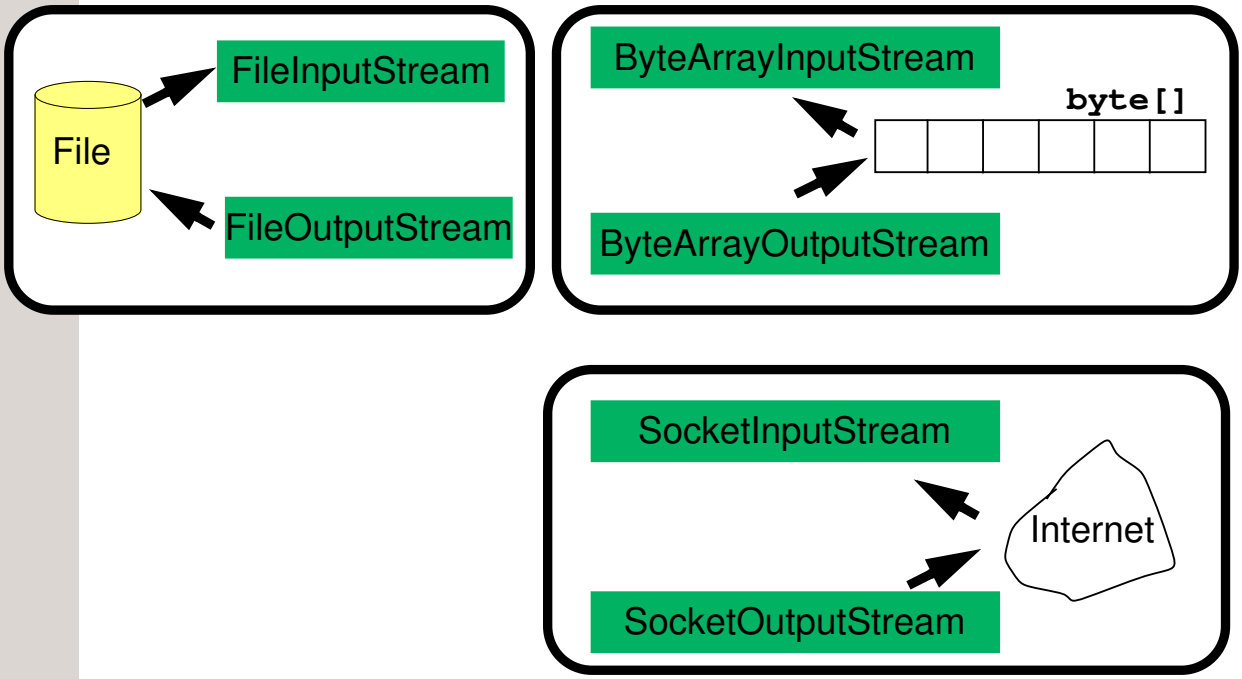
- Where do the data come from / go to?



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.2 Stream Specializations

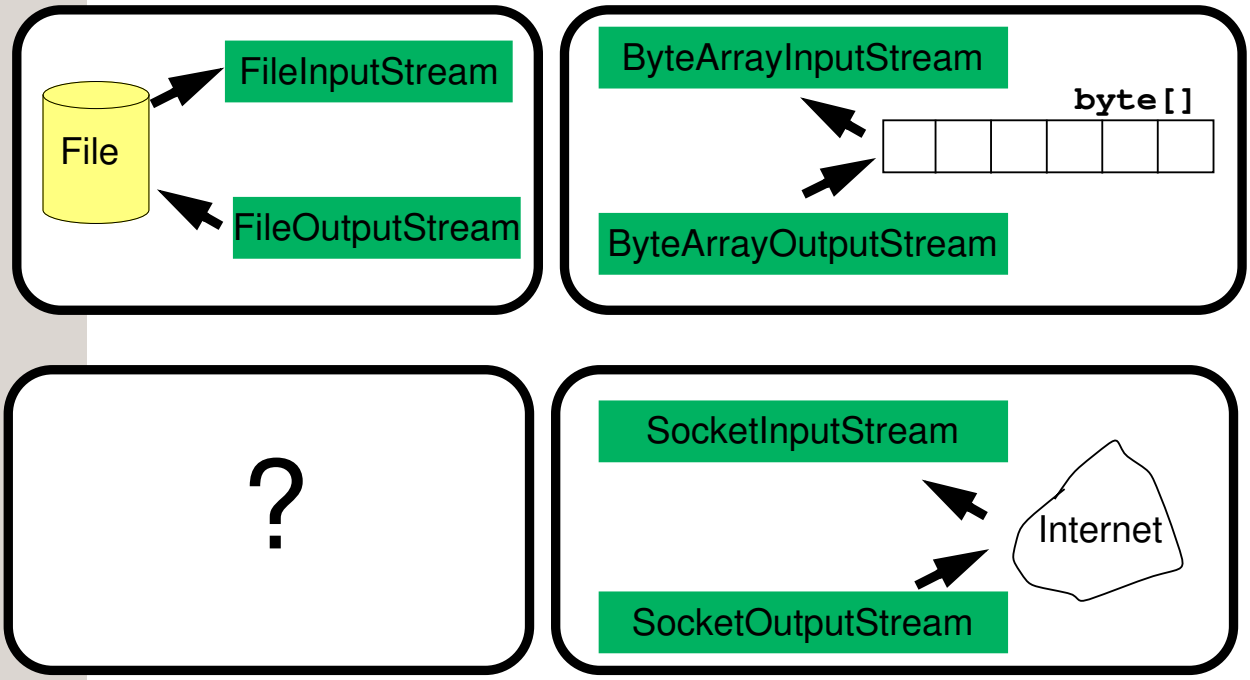
- Where do the data come from / go to?



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

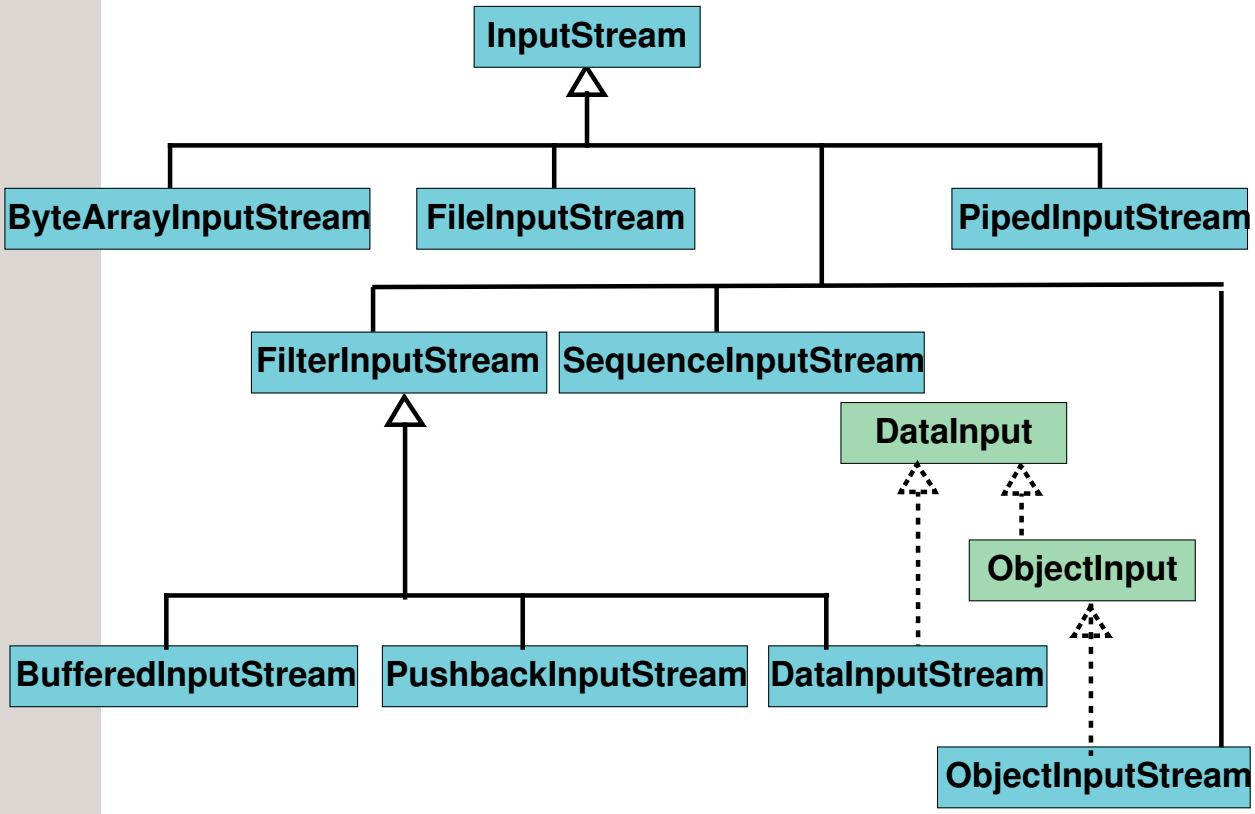
24.2 Stream Specializations

- Where do the data come from / go to?



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.3 Input Streams Class Diagram



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.4 FileInputStream, FileOutputStream

- Read from a file

```
import java.io.*;

public class Test {
  public static void main (String argv[]) throws IOException {
    FileInputStream f = new FileInputStream ("/tmp/test");
    byte buf[] = new byte[4];
    f.read(buf);
  }
}
```

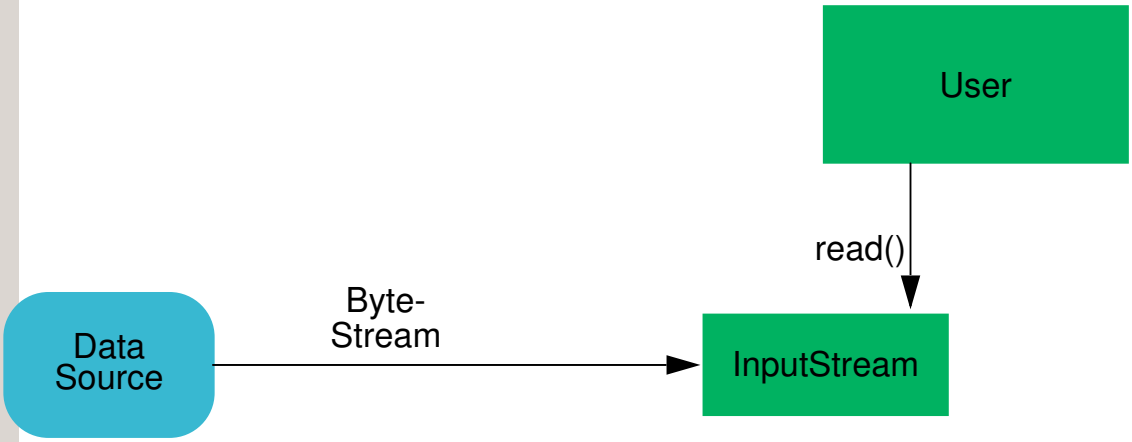

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.5 Combining Streams

- Create comfortable streams from simple streams
- The comfortable stream wraps the simple stream
- ➜ Decorator Design-Pattern

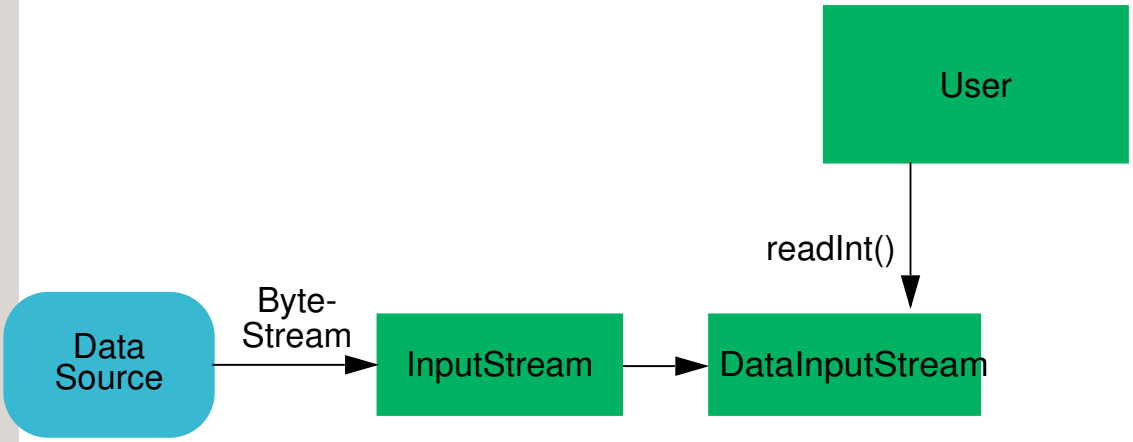
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.6 Combining Streams



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.7 Combining Streams



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.8 DataInputStream

- **`InputStream`** rather uncomfortable
- **`DataInputStream`** used to read *binary representation* of data (int, float,...)
- can be created from every **`InputStream`**

```
InputStream in = new FileInputStream ("/tmp/test");
DataInputStream dataIn = new DataInputStream(in);
float f = dataIn.readFloat();
```

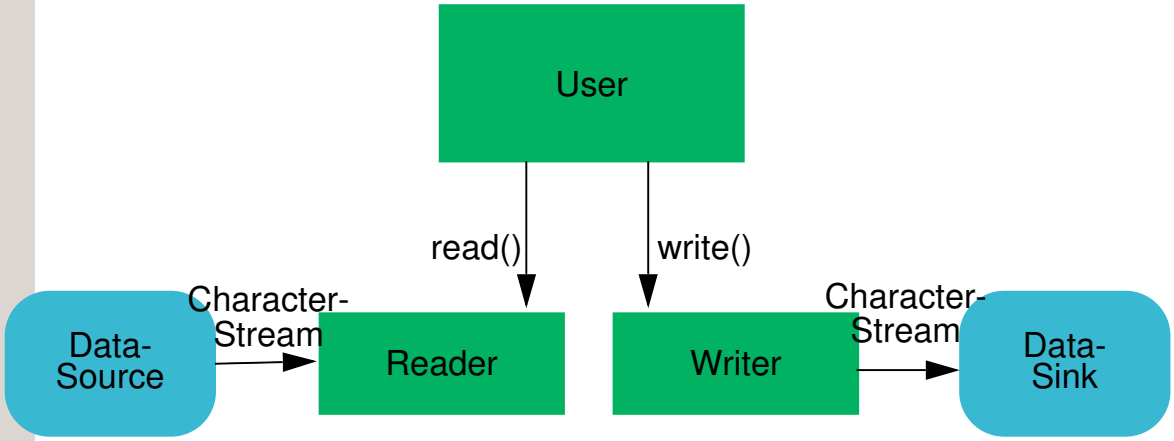
- **`readLine()`** can be used to read whole lines

```
for(;;) {
    String s = dataIn.readLine();
    System.out.println(s);
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.9 Reader/Writer

- Character streams for input/output (**`Reader, Writer`**)



- Character streams contain Unicode characters (16 bit)

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.10 Reader

- important methods:

`int read()`
Read one character and return it as **`int`**.

`int read(char buf[])`
Read characters into buffer. Return number of read characters or -1 in case of error.

`int read(char buf[], int offset, int len)`
Read **`len`** characters in buffer **`buf`** starting at **`offset`**.

`long skip(long n)`
Skip **`n`** characters.

`void close()`
Closes the stream.

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.10.1 FileReader

- Used to read from file
- Constructors:

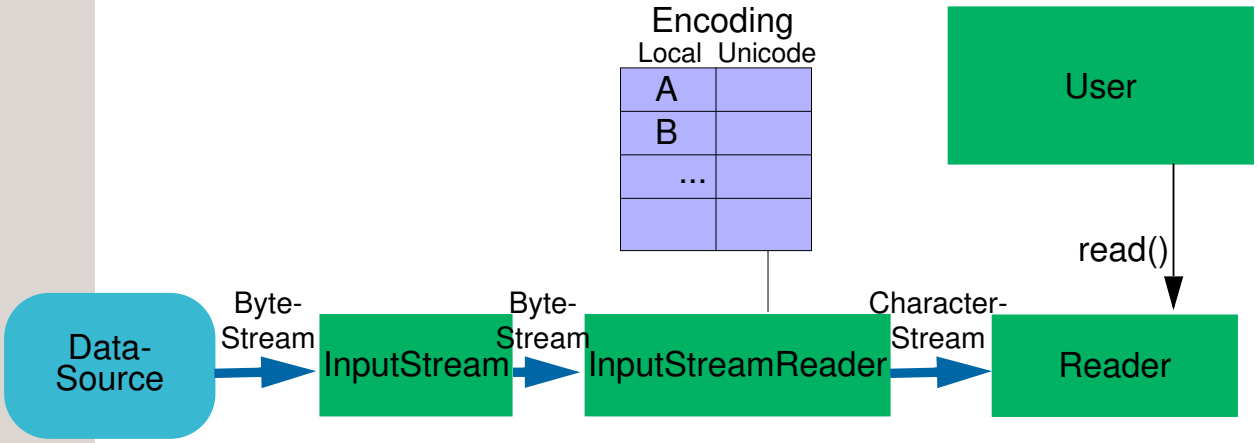
```
FileReader(String fileName)
FileReader(File file)
FileReader(FileDescriptor fd)
```

- No additional methods (only inherited fom **`InputStreamReader`**)
- What is an **`InputStreamReader`**?

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.11 Byte Streams and Character Streams

- Convert byte streams to character streams using an *encoding*



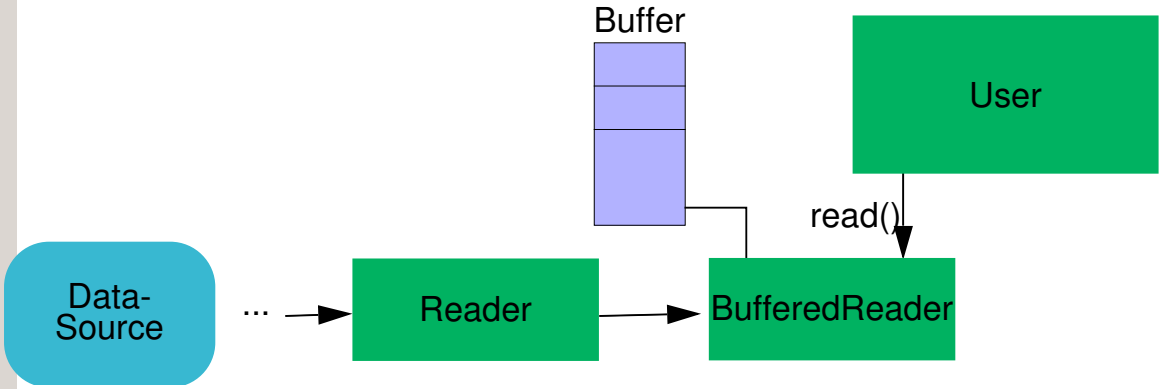
- some encodings: "Basic Latin", "Greek", "Arabic", "Gurmukhi"

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.12 Buffered IO

- Reading/writing single characters can be expensive.
- Converting encodings can be expensive.
- Use **`BufferedReader`**, **`BufferedWriter`** if possible.
- **`BufferedReader`** can be created from every other Reader.
- Important method of **`BufferedWriter`**:

  `void flush():` Empties the buffer - writes buffer to unbuffered writer.



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.13 Buffered IO

- **`BufferedReader`** can read whole lines: **`String readLine()`**

```
BufferedReader in = new BufferedReader(new FileReader("test.txt"));
String line = in.readLine();

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
String line = in.readLine();
```


Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.14 PrintWriter

- Can be created from every **`OutputStream`** or **`Writer`**
- **`println(String s)`**: write string and end-of-line character(s)
- Example: Read file and print it to standard output

```
import java.io.*;

public class CopyStream {
    public static void main(String a[]) throws Exception {
        BufferedReader in = new BufferedReader(
                            new FileReader("test.txt"));
        PrintWriter out = new PrintWriter(System.out);
        for(String line; (line = in.readLine())!=null;) {
            out.println(line);
        }
        out.close();
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

24.15 FileWriter

- used to write characters to a file
- invoke **`close()`** after you finished writing!

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.