Web Technologien

# Exceptions

**Übungsfragen:**

*Was sind Exceptions im Kontext von Java?*

*Vordefinierte Fehlerklassen?*

*Unterschied Exception und Error?*

*Wichtigste Methoden in Klasse Throwable?*

*Aufbau Exceptions?*

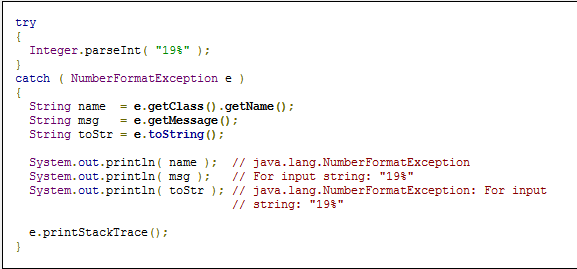
*Konstrukte: try, catch, finally, throw, throws?*

**Warum?** = Abfangen von Fehlern während Laufzeit von Programm

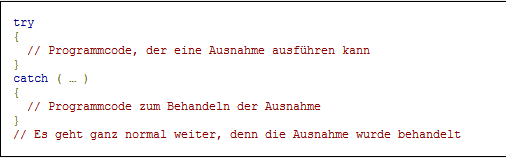
**Was?**

Ausnahmetypen

1. Checked Exception
   * Fehler zur Programmlaufzeit, die behandelt werden müssen, da Programmablauf sonst unterbrochen
2. Unchecked Exception (RunTimeExceptions)
   * Zur Laufzeit nicht behandelt 🡪 Löst keinen Programmcrash aus, nur Teil des Programmablaufs ist fehlerhaft
   * Können optional abgefangen werden
   * Beispiel: IllegalArgumentException, ArrayIndexOutOfBoundException, NullPointerException
3. Error vs. Exception
   * Abnormales Verhalten, auf JVM zurückzuführen
   * Kann zur Dokumentation oder Ausgabe Fehlermeldung abgefangen werden, sonst nicht
   * Beispiel: OutOfMemoryError -> Programm könnte Problem nicht beheben
4. Try Catch Aufbau
   * Try: Ausgeführter Code, der geprüft werden soll
   * Catch: Wirft Exception, Erzeugung eines Objektes, das direkt verarbeitet wird (z.B. Print einer Fehlermeldung) oder an Exception Klasse weitergeworfen wird
     + Informationen über übergebenes Exception Objekt lässt sich über Punktoperator abrufen
     + Werfen verschiedener Exception Objekte über mehrere Catch Blöcke möglich

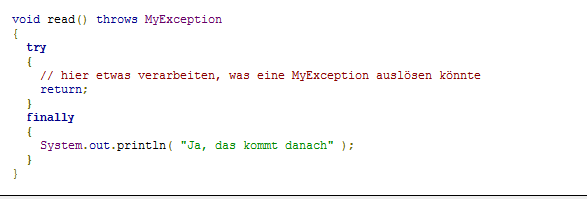


* + Wichtig: Wird ein Fehlerobjekt geworfen, wird der Programmablauf in try unterbrochen, also Code in try Block ab Fehlerauftritt nicht mehr ausgeführt!
    - Möglichkeit, Fehlerbehandlung in Catch Block zu machen und danach Programmablauf weiter auszuführen! (Vgl. Python proceedings mdate & key Tags)
    - Catch Klauseln nach Spezifität aufsteigend anordnen, da sonst bspw. Exception Anweisung jeden Fehler einfangen und weitere Klauseln ignoriert werden würden

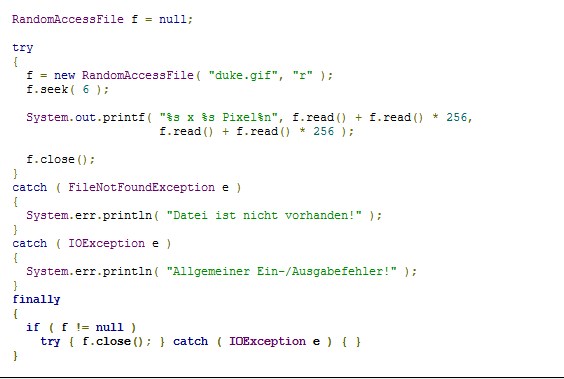


* Stack-Trace
  + Sammeln der Methoden in Stapel, die Fehlerobjekt ausgelöst haben (beginnend mit Methode in try Anweisung), um Fehler nachvollziehen zu können
  + Über printStackTrace() abrufbar
  + Ohne explizites Fehlerabfangen würde zur Laufzeit ebenfalls StackTrace per Default ausgegeben werden, daher try/catch Anweisung für explizite Fehlerbehandlung, sodass Programmlauf nicht unterbrochen!

1. Abschlussbehandlung in „finally“
   * Wird immer ausgeführt, auch wenn keine Exception geworfen wurde!
   * Anwendungsbeispiel: Schließen einer Datei sollte auf jeden Fall ausgeführt werden, unabhängig von Fehlerwurf!
   * Auch bei break, return oder continue in try Block
   * Ausnahme: system.exit() 🡪 Beenden des Programms!
   * Try Block zwingend erforderlich, catch Block nicht



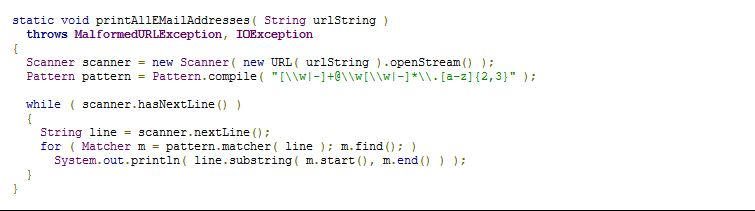
* + - Try immer notwendig, catch optional (dann throws in Methodenkopf deklarieren)



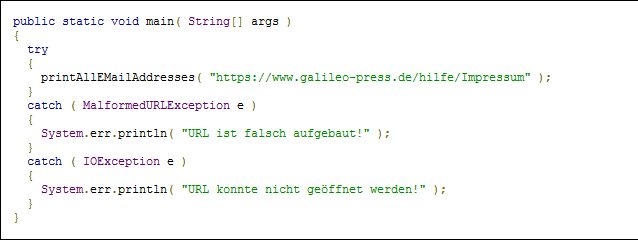
* + - Sollte ein Fehler beim Öffnen oder Auslesen der Datei passieren, würde close() übersprungen werden, daher Ausführung close() in finally Block!

1. Checked Exceptions mit throws im Methodenkopf
   * „Werfen“ eines Fehlers hin zum Aufruf der Methode und dortige Verarbeitung über try/catch
   * Nicht kontrollierte Behandlung von unchecked Exceptions, führen nicht zur Beendigung des Programms!
   * Behandlung von checked Exceptions
   * Programmablauf in aufgerufener Methode wird über throws zu Stelle in main() geworfen, die Methode aufruft und in main() behandelt
   * Verwendung wenn in main() Methoden aufgerufen und Fehler dieser abgefangen werden sollen
   * Weiterleitung und Behandlung in catch Block an anderer Stelle

Weiterleiten über throws in Deklaration der Methode:

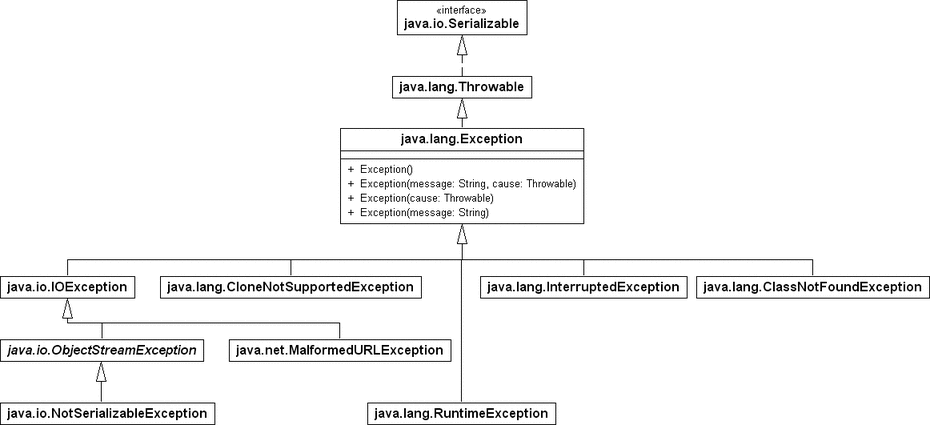


Werfen des Exception Objekts in main() wo Methode aufgerufen wird



* Fehlerweiterleitung an weitere Methoden möglich, sodass Kette an Methodenaufrufen Fehler weitergibt, bis catch Block Fehler behandelt

1. Unchecked Exceptions mit throw behandeln
   * Erzeugung eines Exception Objektes
   * Explizites Auslösen einer Fehlermeldung für Stellen, bei denen Programmablauf nicht gestört werden würde
   * Wird Exception Objekt erzeugt, wird Programmablauf in Methode beendet
   * Benötigt ebenfalls catch Block, der Fehler behandelt
2. Klassenhierarchie bei Exceptions
   * Alle Exceptionklassen erben von Exception, diese von Throwable und diese wiederum von Serializable



* + Problem: Wenn nur oberste Hierarchie an Exceptions gefangen werden würde, könnten verschiedene Ausnahmen in Programmablauf nicht unterschieden, analysiert und separat behandelt werden
  + Beachte: Hierarchisch gleichgestellte Exception Klassen explizit angeben, da sonst nicht über Vererbungsstruktur mitgenommen!

Generics

**Übungsfragen:**

*Welche Vorteile bringen Generics in Java Programmen?*

*Syntax von Generics?*

*Was sind Collections? Was sind bekannte Collections Klassen in Java?*

**Warum?**

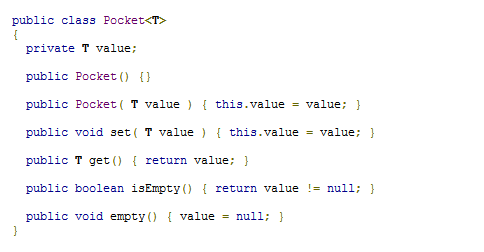
„Platzhalter“ für Deklaration des Datentyps eines Objekts über Typenparameter (Diamondoperator) bei Erzeugung des Objekts, sodass Datentyp flexibel bei neuem Objekt gesetzt werden kann 🡪 kein Cast benötigt bzw. nach Erzeugung des Objekts nachträglich möglich!

Verhindern von ClassCastExceptions zur Laufzeit, wenn Cast nicht möglich

Problem: Nicht ausführbare Casts von Compiler nicht erkannt!

**Was?**

1. Setzen der Platzhalter über Großbuchstaben bei Klassendeklaration die Generics verwenden soll



* + - Ersatz von „Object“ durch „T“ (Typenvariable für generische Typendeklaration)
    - Verwendung anderer Bezeichnungen ungünstig, da Referenz zu Klasse bezogen werden könnte
    - Deklaration mehrerer Typenparameter möglich <T, V>

1. Erzeugung der Objekte und setzen des Datentyps über Typenparameter



* + - Funktionen in Klassen der neuen Objekte die mit generischer Typdeklaration versehen wurden, werden nun alle in Integer bzw. String gecastet

Merke:

* Typenvariablen sind keine Datentypen, sondern Platzhalter

Regeln:

* Typenparameter:
  + E = Elementtyp
  + K = Key
  + N = Nummer
  + T = Typ
  + V = Value
  + S, U, V?

Kurzform der Objekterzeugung mit Generics:

* Leerer Diamond Operator bei “new” statt erneuter Angabe der Typenparameter

Setzen mehrerer Typenparameter:

* Methoden in Klassendeklarationen verschiedenen Datentypen bei Objekterzeugung zuweisen

Generische Methoden:

* Verwendung von Typenparametern in Methoden, um unabhängig von späterer Datentypzuweisung bei Objekterzeugung Vorgänge bearbeiten zu können

Begrenzte Typenparameter:

* Bei Deklaration der Typenparameter kann über „extends“ bestimmter Datentyp spezifiziert werden
  + - Spätere Objekterzeugung erlaubt nur Zuweisung von Datentyp der in Zuweisung erlaubt
    - Beachte: Bei Verwendung in Typendeklaration von Klassen müssen Methoden in Klasse Eingrenzung ebenfalls verwenden

Vererbung bei Generics:

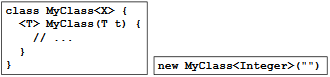
* In <> gesetzte Typenparameter können bei Datentypen Zuweisung in Objekterzeugung keine Vererbungsstrukturen übernehmen
* Beispiel Test<Number> kann bei Objekterzeugung nicht in Test<Integer> zugewiesen werden
* Test<Integer> ist kein Subtyp von Test<Number>!
* Methodenaufruf für erzeugte Instanz kann Vererbungsstruktur (Bspw. Integer von Number) jedoch verwenden!
* Vererbung kann über ‚extends‘ außerhalb spitzer Klammern unter Angabe der Superhierarchie mit Typenparameter in Klassendeklaration ermöglicht werden

Ableiten von Datentypen

* Datentypen von

Generische Konstruktoren:

* Vergabe unterschiedlicher Platzhalter in Klassendeklaration und Konstruktor ermöglichen späteres Setzen verschiedener Datentypen für Argumenttyp in Konstruktor und Datentyp der Klasse



* + - MyClass kann als Integer und mit Konstruktorargument vom Typ String gesetzt werden

Zieldatentypen:

?

Wildcards:

* Verwendung in Spitzen Klammern für unbekannten Datentyp in Parametern, Attribut oder lokaler Variable
* Keine Verwendung in Aufruf als Argument generischer Methoden, Instanziierung generischer Klassen oder Deklarierung des Supertyps

Geschachtelte Klassen

**Übungsfragen:**

*Welche Arten von inneren Klassen gibt es in Java?*

*Vorteile von inneren Klassen?*

*Unterscheidung verschachtelte Klassen und statische Klassen?*

**Warum?**

Verdeutlichung und Erzwingung von Zugehörigkeit von Klassen bei Objekterzeugung

* + - Instanziierung von Objekt innerer Klasse kann nur über Instanziierung von Objekt äußerer Klasse erreicht werden

Beispiel: Organe und Knochen (innere Klassen) können ohne Körper nicht existieren

**Was?**

Definition von Klassen innerhalb anderer Klassen, als „Member“

Innere Klasse hat Zugriff auf Methoden und Variablen äußerer Klasse (selbst wenn Attribute und Methoden auf private gesetzt!)

Keine statischen Members in innerer Klasse möglich, da Änderung von Eigenschaften der äußeren Klasse diese sonst nicht mehr ändern könnten!

Erzeugungsmöglichkeiten des inneren Objekts:

1. Äußeres Objekt instanziieren, dann direkt in main() über Punktoperator auf erzeugtes äußeres Objekt über new ein inneres Objekt erzeugen



1. Methode in äußerer Klasse definieren, die in main() nach Erzeugung des äußeren Objekts inneres Objekt erzeugt



🡪 Definition der Klasse, die Instanziierung der inneren Klasse abarbeitet



🡪 Über Aufruf des zuvor instanziierten äußeren Objekts und der Methode, wird inneres Objekt erzeugt

Methoden von inneren Klassen können nur über instanziierte Objekte der inneren Klasse aufgerufen werden

* **Statische innere Klassen**

Variablen und Methoden werden über Instanz der umliegenden Klasse angesprochen

Instanz der inneren Klasse nur über Methode der äußeren Klasse (return new InnerClass) realisierbar!

# Lokale Klassen

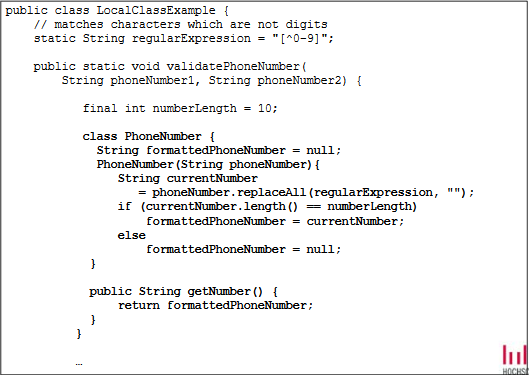
**Warum?**

Abhandlung weiterer Methoden innerhalb einer Methode mit der Instanz eines lokalen Objekts

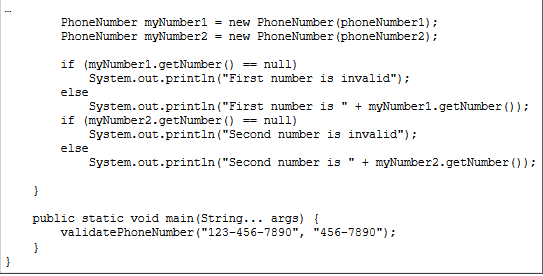
**Was?**

Definition in Methoden

Variablen, Methoden und Instanz der lokalen Klasse definiert



🡪 Deklaration der lokalen Klasse, deren Variablen und Methoden



🡪 Instanziierung der lokalen Klasse in lokaler Klasse und Verarbeitung der lokalen Instanz

(Danach kein Zugriff/Verwendung mehr für Instanz des lokalen Objekts!)

Zugriff auf Members der umgebenden Klasse und Variablen und Parameter innerhalb Methode

Keine statischen Klassen möglich, da Zugriff auf Objekte umgebender Klassen

Keine Interfaces als lokale Klassen möglich, da diese statisch sind

# Anonyme Klassen

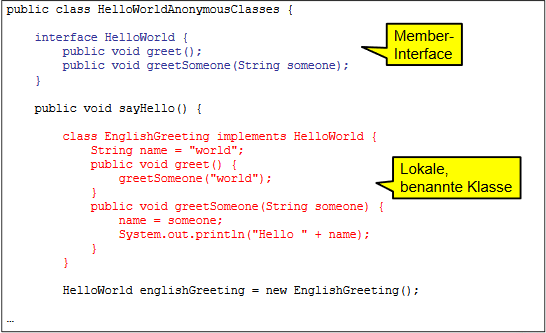
**Warum?**

Alternative bzw. Kurzform der lokalen Klasse 🡪 kompakterer Code

Gleichzeitige Deklaration von Klasse (lokal) und Instanziierung des Objekts

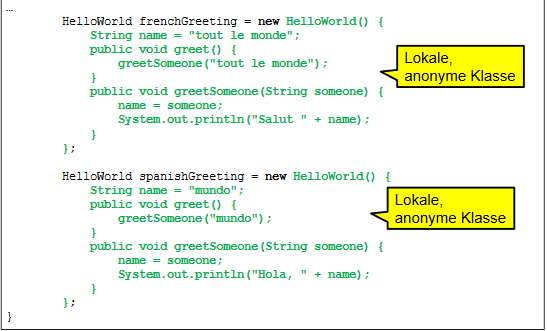
**Was?**

Erzeugung Interface, danach Instanziierung des Interface



🡪Erzeugung des Interface + Deklaration der lokalen Klassen

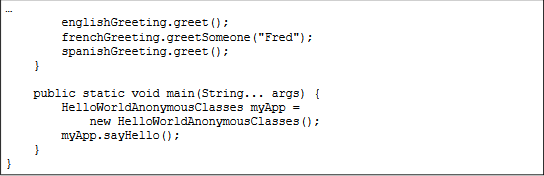
🡪Sieht aus wie Instanziierung, aber inklusive geschweifter Klammer



🡪 frenchGreeting und spanishGreeting als Instanzenvariable direkt instanziiert und deklariert

Merke:

Zugriff auf Instanzen von lokalen oder anonymen Klassen gleich!



Zugriff auf Members der umgebenden Klasse

Zugriff nur auf finale lokale Variablen innerhalb anonymer Klasse, da Klasse nach Verwendung nicht mehr existiert

Interfaces können in anonymer Klasse nicht deklariert werden, da Objekt außerhalb Deklaration nicht mehr verwendet

Keine statischen Members in anonymer Klasse, da Zugriff auf Members der umgebenden Klasse

Verwendung:

Event-Handler für GUIs, da Click auf einen Button Instanz eines Objekts in verschiedenen Ausführungen (Members die unterschiedliche erzeugt

# Aufzählungsdatentypen (Enum)

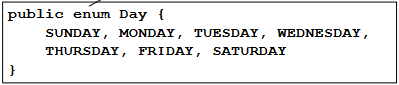
**Warum?**

Mehrfache Instanziierung von selbst kreierten Klassen

**Was?**

Datentyp, der aus „Sammlung“ festgelegter Konstanten besteht (Vgl. Klassenliste)

Deklaration in Körper Datentypdefinition



🡪“Enum“ ersetzt Class in Deklaration

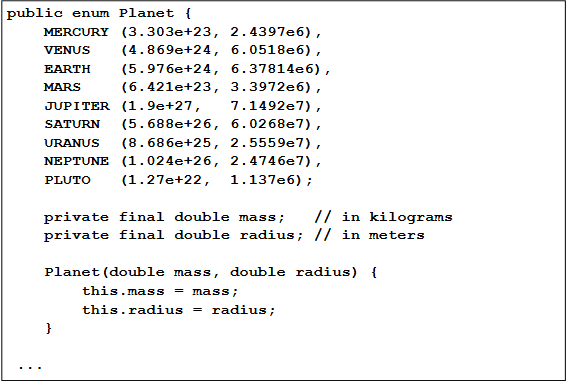
🡪Wochentage sind Instanzen, also „Ausprägungen“ von Day

Merke:

Da bereits Instanzen von Klasse erzeugt wurden, kann „extends“ nicht in Enum Deklaration verwendet werden (Einfachvererbung!)

Methode values() gibt deklarierte Konstanten einer Enum Klasse aus

Enum Klassen können Variablen und Methoden enthalten:



🡪 Variablen als final gesetzt, in Methode/Konstruktor Variablen als Parameter gesetzt und in Konstruktor verwendet

# Servlets

**Warum?**

Übersetzung von explizit übergebenen Daten von Java Code in HTML und umgekehrt, die zwischen Client und Server über http verschickt werden

**Was?**

Instanziierung der Servlet Klassen auf Web Server

Installation von TomCat Server 🡪 Bereitstellung von Servlet API

* **Servlet Lebenszyklus** (Instanziierung bis Beendigung des Servlets)

1. Aufruf von **init()**

🡪 Einmaliger Aufruf der Methode, wenn URL von User adressiert oder Server gestartet wird

🡪 Erzeugt EINE Instanz der Servlet Klasse

🡪 Bei mehrfachem Aufruf gleiche Instanz über Threads angesprochen (Zugriff von verschiedenen Usern, ein Thread pro User)

1. Aufruf von **service(ServletRequest, ServletResponse),** diese ruft GET, POST oder anderen HTTP Request Type pro Thread auf und ruft entsprechende Methode (doGet(), doPost()) auf

🡪 Eine service() Methode pro Request vom Client

🡪 Verarbeitet request und sendet respond zurück an Client

🡪 Kein expliziter Aufruf von Programmierer, sondern Aufruf in Servlet Container, wenn doGet(), usw. von Programm verwendet

1. Aufruf von **doGet(), doPost(), doDelete()**

🡪 Abruf von serverseitigen Daten (von DB, o.ä.) über **request.getParameter()**

🡪 Rückgabe HTML Seite mit erfragten Elementen von Server Seite über **response.getWriter()**

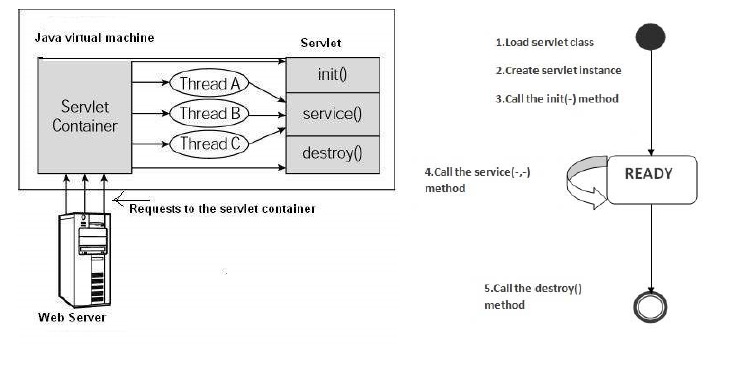
* GET:
  + Daten aus Usereingabe von Client Seite werden an URL angehängt
  + Beschränkt auf 1024 Zeichen
  + Keine Verwendung bei sensible Daten!
* POST:
  + Daten über separate Message versandt

🡪 Daten können über **getParameter(), getParameterValues() und getParameterNames()** vom Server ??? abgerufen werden

1. Aufruf von **destroy()** zum Freigeben der Servlet Instanzfür Garbage Collector

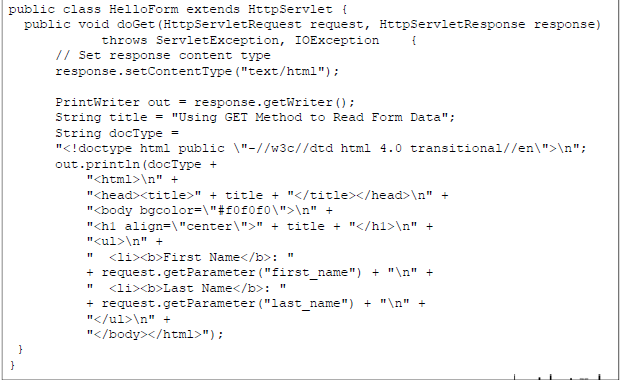
**🡪** Schließen von DB Verbindungen, Beenden von Threads,usw.

* **Architektur des Lebenszyklus**

****

**🡪** Jeder Thread verwendet eigene service() Methode innerhalb EINER Servlet Instanz, ein Thread = ein User

* **doGet /doPost Aufbau**



🡪 Vererbung von **HttpServlet**

🡪 Parameter: HttpServletRequest und HttpServletResponse verarbeiten der Anfrage von Client Seite (dynamische Elemente, hier: Vor- und Nachname) und Antwort von Server Seite (HTML Seite mit dynamischen Elementen, identisch zur Host Page)

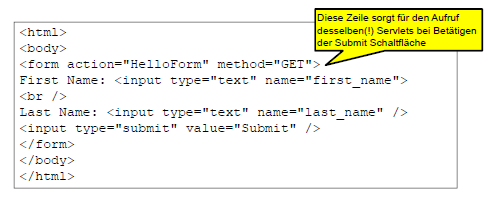
🡪 **response.setContentType(„text/html“)** übermittelt Typ des Dokuments, das von Server zurückgeschickt wird

🡪 **PrintWriter out = response.getWriter()** deklariert Methode die Inhalt der zurückgegeben HTML Seite über **out.println** zurückgibt

🡪 Aufbau der HTML Seite (Doctype Deklaration, <html>, <head>, <body>)

🡪 Abruf der dynamischen Elemente von Serverseite über **request.getParameter(„id“),** id beinhaltet dabei die id des <div> Elements aus HTML Seite

* **Verbindung des HTTP Type Request zur Host Page**

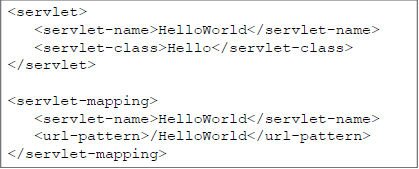


🡪 Über das Setzen von **method=“GET“** in Tag des aufzurufenden HTML Elements ruft in Verbindung mit **action=“HelloForm“** beim Betätigen des Submit Buttons HTML Elemente auf

* **Deployment Descriptor**

Bearbeitung der **web.xml** auf TomCat Server (webapps/ROOT/WEB-INF/web.xml) und in Eclipse (Web-Content/WEB-INF)

Beachte: Unter webapps/ROOT/WEB-INF/classes muss Servlet Klasse abgelegt werden



🡪 **Servlet** beinhaltet **Servlet-name** mit beliebigem Namen für generiertes Servlet und Name der **Servlet Klasse**

🡪 **Servlet-mapping** beinhaltet Servlet-name und **url-pattern** das bei Aufruf im Browser an URL angehängt wird

* **HTTP Anfragen/Antworten**

Zusatzinformationen vom Browser in Header (Key-Value-Paare), die über Methoden von HttpServletRequest und HttpServletRespond abgerufen werden können

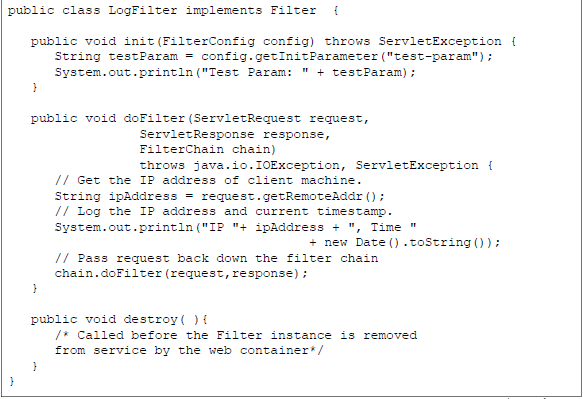
Bsp. Anfragen: Cookies, akzeptierte Sprachen, Host on Port von Client der Anfrage gesendet hat

Bsp. Anworten: Cache Formen (Speicherung des gelieferten Dokuments, Angabe der zeitlichen Überdauerung des Dokuments im Cache)

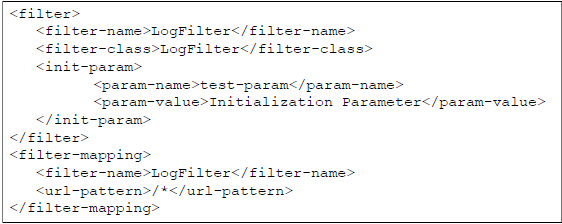
* **Filter**

Eigene Java Klasse zum „Abfangen“ von bestimmten Inhalten aus Anfragen des Clients um diese anderweitig zu verarbeiten oder Modifikation von Antworten des Servers bevor diese an Client geschickt

Einbauen in web.xml, um über Filter-Chain aufgerufen zu werden



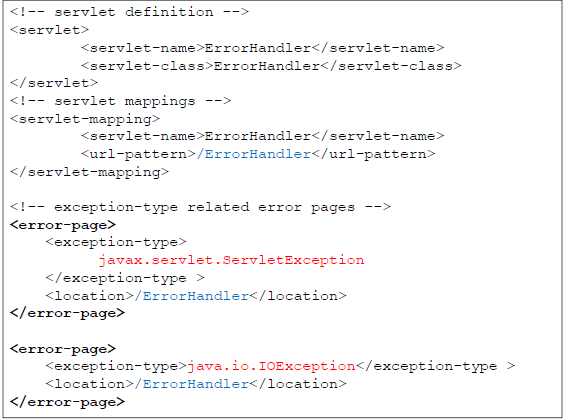
🡪 Methode doFilter soll aus request resultierende IP Adresse an Filter Chain weitergeben



🡪 Einbau von Filtern in web.xml

* **Exceptions für Filter**

Mögliche Fehler: HTTP Error Code (Anfrage von Client nicht erfolgreich bearbeitet) oder Servlet Exception (z.B. IOException)



🡪 Eintrag in web.xml, sodass Servlet aufgerufen (hier ErrorHandler für IOException und ServletException), wenn Fehlertyp in Servlet geworfen

* **Cookies**

Textdateien resultierend aus Eingaben in Browser, die auf Client Rechner gespeichert werden

Werden über HTTP Header verschickt und können über Servlet Methoden aufgerufen werden

Anwendungsbeispiel: Wiederkehrende Nutzer identifizieren

* **Session Tracking**

HTTP Verbindung wird bei jeder Erzeugung neu angelegt und wissen nicht voneinander

🡪 Servlet Klasse kann daher Session Objekt erzeugen, das über bestimmten Zeitraum mehrere HTTP Verbindungen verarbeiten kann

🡪 Timeout des Session Objekts kann über web.xml <session-config> gesetzt werden



1. Mapper (Serverseitige Konzepte, Verbindung DB und Web Server)
2. GWT GUI
3. GWT RPC