

# Objektorientiertes Programmieren in Objective-C unter Mac OS X

19. Mai 2004: Exceptions und System API

# Exceptions

## Was sind Exceptions?

Eine Exception ist eine Ausnahmesituation die den normalen Programmfluß in seiner Ausführung unterbricht. Mit Exceptions kann das Programm diese Situationen kontrollierbar meistern.

## Beispiel:

Dateizugriff mit Rechte-Problemen oder lesen einer nicht vorhandenen Datei  
Netzwerkverbindungen oder Arithmetische Fehler

# Exception Syntax

- Syntax des Exception Handling ähnlich wie in Java und C++
- über Compiler Directive in den Quelltext eingebunden

@try: Quelltext, welche Exception werfen kann

@catch: führt das Exception Handling durch

@throw: wirft eine Exception

@finally: wird immer ausgeführt, ob Exception oder nicht

```
MyClass *my = [[MyClass alloc] init];
```

```
@try {
```

```
    [my anyFunction];
```

```
}
```

```
@catch (NSException *ex){
```

```
    NSLog(@"main: Caught %@: %@", [ex name], [ex  
reason]);
```

```
}
```

```
@finally {
```

```
    [my otherFunction];
```

```
}
```

# Throwing Exception

```
NSError *exception = [NSError exceptionWithName:  
    @"HotTeaException" reason:@"The tea ist too hot" userInfo:nil];  
  
@throw exception;
```

**NSError:**

Klasse von Cocoa Framework zum Exception Handling mit Informationen über die Exception

Auch andere Klassen können über throw als Exceptions geworfen werden: Im Catch-Statement zu beachten! Exception können auch auf höhere Ebene übergeben werden.

```
@try { ... }
```

```
@catch (NSException *ex)  
// notwendige Berechnungen ...  
@throw ex;  
}
```

```
@catch (TUMException *tex){  
    // Code für eigene Ausnahme  
}
```

# Threads

Ein Thread ist die kleinste ausführbare Einheit

Ein Task kann aus mehreren Threads bestehen

Alle Threads nutzen gleichen Namensraum

NSThread für Thread Management unter Cocoa

# System API: Interaktion mit dem Betriebssystem

- **Objective C bietet einige Funktionen um mit dem Betriebssystem zu interagieren**

Starten von Subprozessen

Abfrage der Umgebungsvariablen

Domain Name Lookups

Abfrage des Homedirectory

**NSHost: network name und address informations**

**NSProcessInfo: Prozess weite Informationen**

- übergebene Argumente
- Umgebungsvariablen
- host name
- process name

**NSTask:**

- Ausführen von anderen Programmen
- Überwachen der Programmausführung
- eigene ausführbar Instanz (unterschied zu NSThread )
  - läuft im eigenen Speicherbereich

# Erzeugen und Starten eines NSTask

zwei Möglichkeiten:

In gleicher Umgebung wie aufrufenden Prozess

`LaunchedTaskWithLaunchPath:arguments:`

Erzeugt und startet den Task

wenn Umgebung geändert werden soll: `alloc` und `init` benutzen  
mit den `set`-Methoden die nötigen Umgebungs-Variablen setzen.  
(`setCurrentDirectoryPath`, `setArguments`)

```
- (void)runTask:(id)sender
{
    NSTask *aTask = [[NSTask alloc] init];
    NSMutableArray *args = [NSMutableArray array];
    /* set arguments */
    [args addObject:[inputFile stringValue] lastPathComponent];
    [args addObject:[outputFile stringValue]];
    [aTask setCurrentDirectoryPath:[inputFile stringValue]
        stringByDeletingLastPathComponent];
    [aTask setLaunchPath:[taskField stringValue]];
    [aTask setArguments:args];
    [aTask launch];
}
```

# Beenden eines Tasks

- Beim normalen Programmverlauf endet der ein Task und postet eine *NSTaskDidTerminateNotification* zum standard notification center
- Dabei wird der TerminationStatus bekannt gemacht
- Um darüber informiert zu werden: ein Object als Observer hinzufügen.

```
(id)init {  
    self = [super init];  
  
    [[NSNotificationCenter defaultCenter] addObserver:self  
        selector:@selector(checkATaskStatus:)  
        name:NSTaskDidTerminateNotification  
        object:nil];  
  
    return self;  
}  
  
- (void)checkATaskStatus:(NSNotification *)aNotification {  
    int status = [[aNotification object] terminationStatus];  
    if (status == ATASK_SUCCESS_VALUE)  
        NSLog(@"Task succeeded.");  
    else  
        NSLog(@"Task failed.");  
}
```

# Datenaustausch zwischen Tasks

- Problem des Datenaustausches, da Tasks nicht auf gleichen Adressraum zugreifen können.
- über Filesystem möglich: u.a. Pipes

# Pipes

- Kommunikationsmöglichkeit zwischen 2 Tasks
- muss beim Start eines Tasks angegeben werden

Pipes werden als `NSFileHandle` repräsentiert

`NSFileHandle` dienen zum lesen und schreiben von Daten in die Pipe.

Pipe wird als standard input, standard output und standard error Gerät gesetzt.

```
- (void)readTaskData:(id)sender
{
    NSTask *pipeTask = [[NSTask alloc] init];

    NSPipe *newPipe = [NSPipe pipe];

    NSFileHandle *readHandle = [newPipe fileHandleForReading];

    NSData *inData = nil;

    // write handle is closed to this process

    [pipeTask setStandardOutput:newPipe];

    [pipeTask setLaunchPath:[NSHomeDirectory()
        stringByAppendingPathComponent:@"PipeTask.app/Contents/MacOS/PipeTask"]];

    [pipeTask launch];

    while ((inData = [readHandle availableData]) && [inData length]) {
        [self processData:inData];
    }

    [pipeTask release];
}
```

noch Fragen ?