Lecture 3
summary of Java SE – section 1

presentation
DAD – Distributed Applications Development
Cristian Toma

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics
www.dice.ase.ro
Agenda for Lecture 1 – Summary of JSE

1. Java Exceptions & Source Code Design Patterns
2. IPC & Multi-threading
3. Exchange Ideas & Parallelism Intro
Exception mechanisms and features, source code design patterns – factory methods, singletons

Java Exceptions & Source code design patterns
1.1 Java Exceptions Summary

Propagation Mode:

- Method where error occurred
- Method without an exception handler
- Method with an exception handler
- main

Searching the call stack for the exception handler.

The call stack.
1.1 Java Exceptions Summary

Exceptions Types:

1. *checked exception*
   They are not passing by the compilation phase. May exist a “recovery” mechanism but it is a MUST to have “try-catch” source code statements.

2. *errors*
   They are passing by the compilation phase, BUT it is impossible to forecast malfunctions of HW or OS – e.g. HDD has bad sectors and for opening a file there is a ‘java.io.IOException’ throw. In practice, there is not a try-catch statement for them.

3. *runtime exception*
   They are passing by the compilation phase, BUT the development logics is not implemented correct – e.g. after computations there is a «division by zero». It is possible to use try-catch mechanism but it is better to investigate and to correct the «logic bug».

\[ \text{2 + 3 = unchecked exception} \]
1.1 Java Exceptions Summary

Exceptions Class Hierarchy in Java:
1.1 Java Exceptions Summary

Exceptions C vs. Java/C++ approach:

```java
errorCodeType readFile {
    initialize errorCode = 0;

    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
    } else {
        errorCode = -4;
    }

close the file;
    if (theFileDidntClose && errorCode == 0) {
        errorCode = -5;
    } else {
        errorCode = errorCode and -4;
    }
}

return errorCode;
```
1.2 Java Source Code Design Patterns Summary

Java Singleton:

```
public class SimpleSingleton {
    private static SimpleSingleton singletonInstance = null;
    // Mark the constructor private to avoid object creation outside.
    private SimpleSingleton() {
    }

    // This is where other object can obtain instance of this class.
    public static SimpleSingleton getInstance() {
        if (null == singletonInstance) {
            singletonInstance = new SimpleSingleton();
        }

        return singletonInstance;
    }
}
```

http://searchdaily.net/category/java/designpattern/creational-pattern/
1.2 Java Source Code Design Patterns Summary

Bruce Eckel, “Thinking in Patterns with Java”,
One of the best book for source code design patterns.

Java Singleton:

- “Singleton is used to control the amount of created objects.”
- In same category beside Singleton, there is Objects Pool.

Java Factory Method:

Where to use & benefits
- Connect parallel class hierarchies.
- A class wants its subclasses to specify the object.
- A class cannot anticipate its subclasses, which must be created.
- A family of objects needs to be separated by using shared interface.
- The code needs to deal with interface, not implemented classes.
- Hide concrete classes from the client.
- Factory methods can be parameterized.
- The returned object may be either abstract or concrete object.
- Providing hooks for subclasses is more flexible than creating objects directly.
- Follow naming conventions to help other developers to recognize the code structure.

1.2 Java Source Code Design Patterns Summary


Java Factory Method:

http://searchdaily.net/factory-method-pattern-tutorial/
Java Factory Method:

```java
public class ScientificCalculator extends Calculator {
    public ScientificCalculator(String name) {
        System.out.println("Hello I'm " + name);
    }
}

public class BasicCalculator extends Calculator {
    public BasicCalculator(String name) {
        System.out.println("Hello I'm " + name);
    }
}

public class ProgrammerCalculator extends Calculator {
    public ProgrammerCalculator(String name) {
        System.out.println("Hello I'm " + name);
    }
}
```

http://searchdaily.net/factory-method-pattern-tutorial/
1.2 Java Source Code Design Patterns Summary

Java Factory Method:  
http://searchdaily.net/factory-method-pattern-tutorial/

```java
public class CalculatorFactory {
    public Calculator getCalculator(final String type, final String name) {
        if ("B".equals(type)) {
            return new BasicCalculator(name);
        } else if ("S".equals(type)) {
            return new ScientificCalculator(name);
        } else if ("P".equals(type)) {
            return new ProgrammerCalculator(name);
        } else {
            return new Calculator();
        }
    }
}

public class CalculatorSelector {
    public static void main(String[] args) {
        CalculatorFactory factory = new CalculatorFactory();
        Calculator calculator1 = factory.getCalculator("P", "a Programmer Calculator");
        Calculator c2 = factory.getCalculator("B", "a Basic Calculator");
        System.out.println("c1 type: " + calculator1.getClass().getName());
        System.out.println("c2 type: " + c2.getClass().getName());
    }
}
```

Hello I'm a Programmer Calculator
Hello I'm a Basic Calculator
c1 type: net.searchdaily.java.design.pattern.factorymethod.ProgrammerCalculator
c2 type: net.searchdaily.java.design.pattern.factorymethod.BasicCalculator
Section Conclusion

Fact: DAD needs Java

In few samples it is simple to remember: Exceptions mechanisms and types in Java & source code design patterns such as: Singleton, Objects Pool, Factory Methods...patterns used in any kind of solution – distributed or not.
Linux IPC – Inter-Process Communication, light-weight processes / process thread in C/C++
Linux – pthread library and C++ ’11, JVM and OS threads, Java Multi-threading issues

Linux IPC & Multi-threading in Java & OS Linux
2.1 Summary of MS Windows Memory

Native EXE File on HDD MS Windows:

- EXE File Beginning – ‘MZ’
- EXE 16, 32 bits Headers
- References / pointers to the segments
- Relocation Pointer Table

EXE IMAGE

Firefox

Adobe Reader

Relocation Pointer Table

Optional – Thread 1
Optional – Thread 2
... Optional – Thread n

Ram Memory Layout MS Windows:


EXE IMAGE

Load Module

Firefox

IPC
2.1 Summary of MS Windows Process

IDE Connector

HDD

Binary image of the .COM/.EXE file

Processor x86

Motherboard

BUS (data + instructions)

Interrupt Vector Table

FAT 1 & 2 / NTFS

Interrupt Routines

.PSP

.COM / .EXE Binary Image

Data Segment

STACK

RAM with Virtual Addresses

 KERNEL

OS

0000h

X:0000h

X:0100h
2.1 Summary of Linux/Windows Virtual Memory

2.1 Summary of Linux/Windows Virtual Memory

“Blue regions represent virtual addresses that are mapped to physical memory, whereas white regions are unmapped. In the example above, Firefox has used far more of its virtual address space due to its legendary memory hunger. The distinct bands in the address space correspond to memory segments like the heap, stack, and so on. Keep in mind these segments are simply a range of memory addresses and have nothing to do with Intel-style segments.”
2.1 Summary of Linux/Windows Virtual Memory

MS Windows:

LINUX: http://www.read.cs.ucla.edu/111/2007fall/notes/lec4
2.1 Summary of Linux executable ELF to memory - Process

http://www.cs.umd.edu/~hollings/cs412/s04/proj1/index.html#cast
Before understanding a thread, one first needs to understand a UNIX process. A process is created by the operating system, and requires a fair amount of "overhead". Processes contain information about program resources & program execution state, including:

- Process ID, process group ID, user ID, and group ID;
- Environment;
- Working directory;
- Program instructions;
- Registers;
- Stack;
- Heap;
- File descriptors;
- Signal actions;
- Shared libraries;
- Inter-process communication tools (such as message queues, pipes, semaphores, or shared memory)
2.1 Summary of Processes & IPC in Linux

Processes
- Fork
- Signals

Pipes

FIFO

File-locking

OS Message Queues

Semaphores

Shared Memory

Memory Mapped Files

Sockets
Many people have observed that Linux is difficult for casual users to learn, and that Linux would have a better chance of general acceptance as a desktop platform if it were made easier to use. Pipes are at the root of the great flexibility of Unix, and representing them graphically makes this functionality better accessible to the casual user.
2.1 Summary of IPC in Linux – Fork & Pipes


```c
int p[2];
pipe(p);
```
2.1 Summary of IPC in Linux – Fork & Pipes

Pipe 1: Sending password
'p' 'a' 's' 's' 'w' 'o' 'r' 'd'

Pipe 2: Sending Response
'N'
(password is not a good password)

http://www.read.cs.ucla.edu/111/_media/notes/ipc_pipes_1.gif
2.1 Summary of IPC in Linux – Message Queues

Linux C - System V API / POSIX API

## 2.1 Summary of IPC in Linux – Message Queues

**Linux C - System V API / POSIX API**

<table>
<thead>
<tr>
<th>Operation</th>
<th>POSIX Function</th>
<th>SVR4 Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gain access to a queue, creating it if it does not exist.</td>
<td>mq_open(3)</td>
<td>msgget(2)</td>
</tr>
<tr>
<td>Query attributes of a queue and number of pending messages.</td>
<td>mq_getattr(3)</td>
<td>msgctl(2)</td>
</tr>
<tr>
<td>Change attributes of a queue.</td>
<td>mq_setattr(3)</td>
<td>msgctl(2)</td>
</tr>
<tr>
<td>Give up access to a queue.</td>
<td>mq_close(3)</td>
<td>n.a.</td>
</tr>
<tr>
<td>Remove a queue from the system.</td>
<td>mq_unlink(3),</td>
<td>msgctl(2),</td>
</tr>
<tr>
<td></td>
<td>rm(1)</td>
<td>ipcrm(1)</td>
</tr>
<tr>
<td>Send a message to a queue.</td>
<td>mq_send(3)</td>
<td>msgsnd(2)</td>
</tr>
<tr>
<td>Receive a message from a queue.</td>
<td>mq_receive(3)</td>
<td>msgrcv(2)</td>
</tr>
<tr>
<td>Request asynchronous notification of a message arriving at a</td>
<td>mq_notify(3)</td>
<td>NA</td>
</tr>
<tr>
<td>queue.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

http://menehune.opt.wfu.edu/Kokua/More_SGI/007-2478-008/sgi_html/ch06.html
2.2 Summary of Multi-threading in C vs. Java

Multi-threading vs. Multi-process development in UNIX/Linux:

https://computing.llnl.gov/tutorials/pthreads/
2.2 Summary of Multi-threading in C vs. Java

Threads Features:

Thread operations include thread creation, termination, synchronization (joins, blocking), scheduling, data management and process interaction.

A thread does not maintain a list of created threads, nor does it know the thread that created it.

All threads within a process share the same address space.

Threads in the same process share:
- Process instructions
- Most data
- open files (descriptors)
- signals and signal handlers
- current working directory
- User and group id

Each thread has a unique:
- Thread ID
- set of registers, stack pointer
- stack for local variables, return addresses
- signal mask
- priority
- Return value: errno

pthread functions return "0" if OK.
2.2 Summary of Multi-threading in C vs. Java

Multi-threading in C/C++ with pthread (Is “counter++” an atomic operation?):

```c
Without Mutex
1: int counter=0;
2: /* Function C */
3: void functionC()
4: {
5:     counter++
6: }

With Mutex
01: /* Note scope of variable and mutex are the same */
02: pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
03: int counter=0;
04: /* Function C */
05: void functionC()
06: {
07:     pthread_mutex_lock( &mutex1 );
08:     counter++
09:     pthread_mutex_unlock( &mutex1 );
10: }
```

Possible execution sequence:

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter = 0</td>
<td>counter = 0</td>
<td>counter = 0</td>
<td>counter = 0</td>
</tr>
<tr>
<td>counter = 1</td>
<td>counter = 1</td>
<td>counter = 1</td>
<td>Thread 2 locked out.</td>
</tr>
<tr>
<td>Thread 1 has exclusive use of variable counter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>counter = 2</td>
<td></td>
<td></td>
<td>counter = 2</td>
</tr>
</tbody>
</table>
2.2 Summary of Multi-threading in C vs. Java

Multi-threading vs. Multi-process mini-terms:

**Mutexes** are used to prevent data inconsistencies due to race conditions.

A race condition often occurs when two or more threads need to perform operations on the same memory area, but the results of computations depends on the order in which these operations are performed.

**Mutexes** are used for serializing shared resources. Anytime a global resource is accessed by more than one thread the resource should have a Mutex associated with it.

One can apply a mutex to protect a segment of memory ("critical region") from other threads.

**Mutexes** can be applied only to threads in a single process and do not work between processes as do semaphores.

In Java Mutex is quite ↔ synchronized
2.2 Summary of Multi-threading in C vs. Java

Multi-threading Models:

http://www-sop.inria.fr/indes/rp/FairThreads/FTJava/documentation/FairThreads.html#One-one-mapping
2.2 Summary of Multi-threading in C vs. Java

Multi-threading Models:

http://www-sop.inria.fr/indes/rp/FairThreads/FTJava/documentation/FairThreads.html#One-one-mapping

Many-to-one Mapping

Many-to-many Mapping
2.2 Summary of Multi-threading in C vs. Java

JVM Multi-threading Model mapping to OS native threads:

- Standard N-M Thread Model

  - JVM
  - User Space
  - Native – C/C++
  - Kernel Space

Java Application
Solaris Libthread
Solaris Kernel

- Java Thread
- Solaris Thread
- LWP
- Kernel Thread

http://192.9.162.55/docs/hotspot/threads/threads.html
2.2 Summary of Multi-threading in C vs. Java

JVM Multi-threading Model mapping to OS native threads:

http://www.javamex.com/tutorials/threads/how_threads_work.shtml

Inside the Java Virtual Machine, Bill Venners
### 2.3 Summary of Multi-threading in Java

#### Java Multi-threading API

<table>
<thead>
<tr>
<th>Option 1 – Java API for Multi-Threading Programming</th>
<th>Option 2 – Java API for Multi-Threading Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Defining the classes:</strong></td>
<td><strong>Defining the classes:</strong></td>
</tr>
<tr>
<td>class Fir extends Thread {</td>
<td>class Fir extends Ceva implements Runnable {</td>
</tr>
<tr>
<td>public void run() {...}</td>
<td>public void run() {...}</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td><strong>Instantiates the objects</strong></td>
<td><strong>Instantiates the objects</strong></td>
</tr>
<tr>
<td>Fir f = new Fir();</td>
<td>Fir obf = new Fir();</td>
</tr>
<tr>
<td></td>
<td>Thread f = new Thread(obf);</td>
</tr>
<tr>
<td><strong>Set the thread in ‘Runnable’ state</strong></td>
<td><strong>Set the thread in ‘Runnable’ state</strong></td>
</tr>
<tr>
<td>f.start();</td>
<td>f.start();</td>
</tr>
<tr>
<td><strong>Specific Thread methods calls</strong></td>
<td><strong>Specific Thread methods calls</strong></td>
</tr>
<tr>
<td>public void run() {</td>
<td>public void run() {</td>
</tr>
<tr>
<td>Thread.sleep(...);</td>
<td>Thread.sleep(...);</td>
</tr>
<tr>
<td>String fName = this.getName();</td>
<td>Thread t = Thread.currentThread();</td>
</tr>
<tr>
<td>...</td>
<td>String fName = t.getName();</td>
</tr>
<tr>
<td>}</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
2.3 Summary of Multi-threading in Java

Java Thread States

- new thread
- call start
- runnable thread
- thread completes
- blocked thread
- dead thread
- call sleep
  - call wait
  - blocking I/O request
- sleep time ends
  - call notify or notify all
  - I/O complete
2.3 Summary of Multi-threading in Java

Java Multi-threading Synchronization for Concurrent Programming

Sincronizarea firelor de executie:

Problema sincronizarii apare in cazul de:
- A. Concuranta (se incearca utilizarea acelorasi resurse)
- B. Cooperare (se incearca interschimb de informatii)

A. Rolul sincronizarii in acest caz este de a asigura accesul exclusiv la resursele comune. O secventa de cod pentru care trebuie sa se execute accesul exclusiv (un singur proces/fir de executie executa la un moment dat secventa respectiva de cod) se numeste regiune critica.

A1. Mai multe fire de executie pot modifica acceasi resursa comuna (FirNe2.java => FirSi2.java)

A2. Mai multe fire de executie pot executa in paralel metodele aceluiasi obiect (FirNe3.java => FirSi3.java)

* Aparent similar dar cu rezolvari diferite.

B. Sincronizarea inseamna in acest caz ca procesele (firele de executie), care se pot executa cu viteze diferite, trebuie sa se astepte unele pe altele pana cand pot sa-si transmita informatiile de care au nevoie.
2.3 Summary of Multi-threading in Java

Java Multi-threading Synchronization for Producer-Consumer

ProducatorConsumator

Probleme speciale:

- Metodele `wait()`, `notify()` și `notifyAll()` sunt a clasei `Object` și nu a clasei `Thread`. Metoda `wait()` poate fi apelată DOAR dintr-o metoda sincronizată.
- De multe ori în programare este necesar să se asociei unui zavor mai multe conditii/evenimente care ar putea să producă blocarea unor fire de executie care utilizează obiectul respectiv (resursa comuna). Se pune problema cum se face deosebirea între aceste evenimente. Solutia consta din asocierea unor conditii logice la evenimentele respective. O metoda `wait()` se apeleaza intr-un ciclu de forma:

```java
while (!conditie) {
    this.wait();
}
```

- Pentru variable ce formeaza resursa comuna se recomanda utilizarea cuvantului cheie `volatile`. Atributul este folosit pentru a indica faptul că o variabila poate sa-si modifice valoarea datorita unei metode nesincronizate. Orice modificare trebuie scrisa cat mai repede in memorie si nu pastrata intr-un registru in microprocesor pentru optimizare.
2.3 Summary of Multi-threading in Java

Java Multi-threading Cooperation

- Daca un alt fir de executie (inclusiv cel pe care se executa metoda void main(String[] args)) trebuie sa astepte un alt fir de executie poate utiliza metoda `join()`:

```java
class Fir extends Thread {
    private double rezultat;

    public void run() { this.rezultat = this.calculeaza(); }

    public double calculeaza() {
        double valoare;
        //... o gelenie de calcule
        return valoare;
    }

    public double getSolutie() { return this.rezultat; }
}

class ProgMainFir1 {
public static void main(String[] args) {
    Fir f1 = new Fir();
    Fir f2 = new Fir();

    f1.start();
    f2.start();
}
}

class ProgMainFir2 {
public static void main(String[] args) {
    Fir f1 = new Fir();
    Fir f2 = new Fir();

    f1.start();
    Thread current = Thread.currentThread();
    f2.setPriority(current.getPriority() + 1);
    f2.start();
}
```
2.3 Summary of Multi-threading in Java

Java Multi-threading Cooperation

- Metoda `yield()` permite sistemului de operare să omoare controlul de la firul de executie curent la un alt fir de executie cu aceeasi prioritate:

```java
// modificarea metodei run() pentru exemplu anterior
public void run() {
    while (true) {
        System.out.println(mesaj);
        this.yield(); // daca firele de executie au aceeasi prioritate, vor alterna
    }
}
```

- Pentru setarea prioritatilor (`setPriority()` si `getPriority()` ) trebuie rulat exemplul `Priority.java`
- Pentru exemplu de utilizare metodele `join()`, `join(...)`, `isAlive()` si `interrupt()` trebuie rulat `SimpleThreads.java`

Java MUTEX ↔ `synchronized` * poate fi folosit la nivel de metoda daca si numai daca metoda face parte dintr-o clasa care NU este derivata din Thread (implementeaza Runnable)

Care e diferenta intre semafor si variabile mutex?

Ce obiecte/instante sunt thread-safe? – immutable, singleton, “normale”, “finale”?
Section Conclusions

All threads in a program must run the same executable. A child process, on the other hand, may run a different executable by calling an exec function.

An errant thread can harm other threads in the same process because threads share the same virtual memory space and other resources. For instance, a wild memory write through an uninitialized pointer in one thread can corrupt memory visible to another thread.

An errant process, on the other hand, cannot do so because each process has a copy of the program’s memory space.

Copying memory for a new process adds an additional performance overhead relative to creating a new thread. However, the copy is performed only when the memory is changed, so the penalty is minimal if the child process only reads memory.

Threads should be used for programs that need fine-grained parallelism. For example, if a problem can be broken into multiple, nearly identical tasks, threads may be a good choice. Processes should be used for programs that need coarser parallelism.

Sharing data among threads is trivial because threads share the same memory. However, great care must be taken to avoid race conditions. Sharing data among processes requires the use of IPC mechanisms. This can be more cumbersome but makes multiple processes less likely to suffer from concurrency bugs.
Share knowledge, Empowering Minds

Communicate & Exchange Ideas
Some “myths”:

(Distributed Systems).Equals(Distributed Computing) == true?

(Parallel System).Equals(Parallel Computing) == true?

(Parallel System == Distributed System) != true?

(Sequential vs. Parallel vs. Concurrent vs. Distributed Programming) ? (Different) : (Same)

if (HTC != HPC)
    HTC (High Throughput Computing) >
    MTC (Many Task Computing) >
    HPC (High Performance Computing);

... Will be continued! - In the next lectures ...
Flynn Taxonomy
Parallel vs. Distributed Systems

Parallel vs. Distributed Computing / Algorithms

http://en.wikipedia.org/wiki/Flynn's_taxonomy
http://en.wikipedia.org/wiki/Distributed_computing

Where is the picture for: Distributed System and Parallel System?
Parallel Computing & Systems - Intro
https://computing.llnl.gov/tutorials/parallel_comp/

Serial Computing

Parallel Computing
Parallel Computing & Systems - Intro
https://computing.llnl.gov/tutorials/parallel_comp/
Parallel Computing & Systems - Intro
https://computing.llnl.gov/tutorials/parallel_comp/

Top500 HPC Application Areas
Assignment 03 – Use (All of them – logic AND):
- C/C++ POSIX Thread (pthread) using gcc;
- C++’11 Multi-threading programming using gcc;
- Java Multi-threading with JDK

in Ubuntu 12 (download the virtual machine from http://acs.ase.ro), in order to add item by item, 2 (two) extra large vectors in the third vector as result. The sum of each segment/chunk should take place in parallel on each core of the microprocessor/(s) – for details check-out SAKAI and http://acs.ase.ro.
Thanks!

DAD – Distributed Application Development
End of Lecture 3 – summary of Java SE – section 1