Threads
Overview

- support for multi-threaded applications in the language
- “main” thread of an application – the main() method
- in JVM there are always a number of threads
  - depends on the implementation
- JVM terminates after termination of all threads (which are not daemon threads)
- threads and thread groups
- support for synchronization
  - synchronized
Thread creation

- thread implementation
  1. extending the class `java.lang.Thread`
  2. implementing the interface `java.lang.Runnable`

- extending the `Thread`
  - redefining the method `void run()`
  - the thread is started by the method `start()`

- interface `Runnable`
  - the only method `void run()`
  - implemented by a class
  - the thread start – `new Thread(Runnable).start()`
public class SimpleThread extends Thread {
    public SimpleThread() {
        start();
    }
    public void run() {
        for (int i=0; i<5; i++)
            System.out.println(getName() + " : "+i);
    }
    public static void main(String[] args) {
        for (int i=0; i<5; i++) {
            new SimpleThread();
        }
    }
}
**yield**

- method of the class Thread
  - temporarily suspending the thread in order another thread can run
  - it is only a recommendation
- static method
- update of the previous example

```java
public void run() {
    for (int i=0; i<5; i++) {
        System.out.println(getName() + " : "+i);
        yield();
    }
}
```
sleep

- two methods of the Thread
  - `sleep(int milis)`
  - `sleep(int milis, int nanos)`
    - nanos within range 0-999999
- static method
- causes the currently executing thread to sleep for the given time
- can be interrupted (by the method `interrupt()`)
  - throws the exception `InterruptedException`
• interrupts “waiting” of a thread
• three methods of the class Thread
  – `join()`
    • waits for the given thread to terminates
  – `join(int milis)`
  – `join(int milis, int nanos)`
    • waits for the given thread to terminates but at most the given time (0..as `join()` without parameters)
• can be interrupted
  – exception `InterruptedException`
Priority

- each thread has the priority
- void setPriority(int newPriority)
- int getPriority()
- 10 levels
- constants
  - MAX_PRIORITY = 10
  - MIN_PRIORITY = 1
  - NORM_PRIORITY = 5
- groups of threads (ThreadGroup)
  - getMaxPriority()
  - setPriority()
    - it sets the priority only up to the max priority for the group to which the thread belongs
Daemon threads

- "management" threads
- runs "in background"
- they do not directly belong to an application
  - e.g. the thread for garbage collector
- JVM terminates after termination of all non-daemon threads
- methods
  - void setDaemon(boolean daemon)
    - can be called on not-yet-started thread only
  - boolean isDaemon()
Synchronization

- there is a lock associated with each instance
- there is a lock associated with each class

- command/modifier synchronized

- command
  - synchronized (expression) Block
  - expression must evaluate to a reference
  - before the Block is to be executed, the thread must obtain the lock on the instance specified by the expression
  - after the Block is finished, the lock is released
Synchronization

• modifier of a method
  - synchronized in the signature of the method
  - behaves in the same manner like the command synchronized
  - the thread also before execution of the method must obtain the lock on the instance
  - after the method is finished, the lock is released
  - static synchronized methods manipulates the lock associated with the class

• mutually excluded are only synchronized methods and blocks

• if a thread has obtained the lock on an instance – other threads can use fields of the instance and call non-synchronized methods of the instance
wait & notify

• there is a queue of waiting threads associated with each instance
  – it is empty after creating the instance
• it is used by the methods wait, notify and notifyAll
  – defined in java.lang.Object
• void wait()
  – can be called only when the calling thread has obtained the lock on the given instance (i.e. in a synchronized section)
    • or throws the exception IllegalMonitorStateException
  – puts the thread to the queue of waiting threads, and
  – releases the lock on the instance
    • other threads can obtain the lock, i.e. enter synchronized sections
**wait & notify**

- the thread is in the queue of waiting threads until the `notify` or `notifyAll` method is called
- `void notify()`
  - "wakes up" a thread from the queue (if the queue is not empty)
  - can be called only from synchronized sections
    - jinak výjimka IllegalMonitorStateException
  - the waked up thread continues after it obtains the lock (i.e. after the tread, which held the lock (and called notify) leaves the synchronized section)
- `void notifyAll()`
  - "wakes up" all threads from the queue
  - the threads can continue after they obtain the lock
wait & notify

• three wait methods
  – void wait()
  – void wait(int milis)
  – void wait(int milis, int nanos)
    • threads stay in the queue till waked up or the given time has elapsed
• waiting in the wait() can interrupted (the method interrupt())
  – the exception InterruptedException is thrown
• wait, notify, and notifyAll are final

• the method sleep() does not releases the lock
public class SimpleLock {

    private boolean locked;

    public SimpleLock() {
        locked = false;
    }

    synchronized public boolean lock() {
        try {
            while (locked)
                wait();
            locked = true;
        } catch (InterruptedException e) {
            return false;
        }
        return true;
    }

    synchronized public void unlock() {
        locked = false;
        notify();
    }
}
Stopping thread

- `destroy()`
- `stop()`
- `stop(Throwerable t)`
- `suspend()`
- `resume()`
  - all of them *deprecated* (most since JDK 1.2)
  - dangerous
  - can cause an inconsistent state of an application or cause a deadlock

- `destroy()` and `stop(Throwerable)`
  - removed since Java 11
Thread groups

- a thread can belong to a group of threads
- the ThreadGroup class
- a group can contain threads and other groups
  - tree hierarchy
- can be obtained
  - all threads in the group
  - parent group in the hierarchy
  - active threads in the group

- can be ignored
Thread name

- each thread has a name
  - can be specified during creation
    - constructors
      - Thread(String name)
      - Thread(Runnable obj, String name)
  - after creation
    - setName(String name)
  - obtaining the name
    - String getName()

- if the name is not set, then it is assigned automatically
  - "Thread-"+n
  - n is sequence number
Other methods

- static Thread currentThread()
  - returns a reference to the currently executing thread
- boolean isAlive()
  - test if this thread is alive
    - false in case the thread is not yet started or already finished
- boolean isInterrupted()
  - test whether this thread has the flag interrupted assigned
- boolean interrupted()
  - as isInterrupted(), but clears the flag interrupted
- String toString()
  - the string contains
    - name
    - priority
    - group
java.util.concurrent

- java.util.concurrent
- java.util.concurrent.atomic
- java.util.concurrent.locks

- since JDK 5
- contain classes for concurrent access to data, locks, semaphores,...
java.util.concurrent

• Executor
  – interface
  – multiple implementations
    • ThreadPoolExecutor, ForkJoinPool, ...
  – void execute(Runnable command)
    • executes the “command” at some time in future

• ExecutorService
  – interface, extends Executor
  – additional methods
    • Future<T> submit(Callable<T> task)
    • List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)
    • ...

Java, winter semester 2018
19. 11. 2018
java.util.concurrent

- Callable<T>
  - interface
  - T call() throws Exception
  - equivalent to Runnable, but returns a value and can throw an exception

- Future<T>
  - interface
  - a result of an asynchronous operation
  - T get()
    - returns the result
    - waits if the result is not yet available
ForkJoinPool

- od Java 7
- implements ExecutorService
- intended for “divide and conquer”
- supports “work-stealing”

- ForkJoinTask<V>
  - a task for ForkJoinPool, an abstract class
  - children
    - RecursiveAction
      - abstract void compute()
    - RecursiveTask<V>
      - abstract V compute()
ForkJoinPool

- methods for executing tasks
  - execute()
    - asynchronous execution
  - submit(), submitAll()  
    - asynchronous execution + returns a Future
  - invoke(), invokeAll()
    - execution and waiting for a result

- similar methods are also in ForkJoinTask
  - execution of “subtasks”

- obtaining the pool
  - constructors, or
  - ForkJoinPool.commonPool()
class CustomRecursiveAction extends RecursiveAction {

    @Override
    protected void compute() {
        if (...) {
            ForkJoinTask.invokeAll(createSubtasks());
        }
    }

    public static void main() {
        CustomRecursiveAction cra =
                new CustomRecursiveAction()
                ForkJoinPool.commonPool().invoke(cra);
    }
}
Executors

- a class
- only static utility methods
  - converting Runnable into Callable
  - obtaining different thread-pools
    - newFixedThreadPool()
java.lang.System
java.lang.System

- contains static elements only
- no instance can be created

- fields
  - java.io.InputStream in
    - standard input
  - java.io.PrintStream out
    - standard output
  - java.io.PrintStream err
    - standard error output
Metods

- `void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
  - copies arrays
  - works even if `src==dest`

- `long currentTimeMillis()`
  - current time in milliseconds since 1.1.1970
  - precision depends on OS

- `long nanoTime()`
  - value of a system timer in nanoseconds
  - nanoseconds since some fixed but arbitrary time
    - can even in future, i.e. the returned value can negative
  - used for measurements of time intervals
  - since Java 5
Metods

- **void exit(int status)**
  - terminates JVM
- **void gc()**
  - recommendation for JVM to run garbage collector
- **void setIn(InputStream s)**
  **void setOut(PrintStream s)**
  **void setErr(PrintStream s)**
  - sets the particular input/output
- **int identityHashCode(Object x)**
  - returns default hash code of the object
Properties

- tuples
  - key – value
  - String (both keys and values)
- system and user-defined
- `Properties getProperties()`
  - returns all set properties
  - `java.util.Properties` – extends `java.util.Hashtable`
- `String getProperty(String key)`
  - returns the value
  - if the key is not set, returns `null`
- `String getProperty(String key, String def)`
  - returns the value
  - if the key is not set, returns `def`
Properties

- void setProperties(Properties props)
  - sets properties in props
- String setProperty(String key, String val)
  - sets the given property property
  - returns its previous value or null
- String clearProperty(String key)
  - clears the given property

- setting properties at JVM start
  - parameter -Dkey=value
  - ex. java -DdefaultDir=/usr Program
- typically, hierarchical names (separated by dots) are used as the keys
Always set properties

- java.version
- java.home
  - directory where the Java is installed
- java.class.path
- java.io.tmpdir
  - directory for temporary files
- os.name, os.architecture, os.version
  - identification of an operating system
- file.separator
  - the separator of names in a path (unix "/", win "\\")
- path.separator
  - the path separator (unix ":", win ";")
- line.separator
  - the line separator (unix "LF", win "CR LF")
Always set properties

- user.name
  - name of the current user
- user.home
  - user's home dir
- user.dir
  - current directory
- plus several properties that identifies VM
Environment variables

- Map\(<String, String>\) `getenv()`
  - all set environment variables
  - unmodifiable collection
- `String getenv(String name)`
  - variable with the given name
java.lang.Runtime
Runtime

• there is always a single instance
  - no other instances can be created
• Runtime getRuntime()
  - static method
  - returns the instance of the Runtime
• int availableProcessors()
  - depends on the implementation
  - returned value may change during a program execution
• long freeMemory()
  - free memory available for JVM
• long maxMemory()
  - maximal available memory for JVM
• void halt(int status)
  - immediately terminates JVM, does not wait for anything
**Runtime**

- **void addShutdownHook(Thread hook)**
  - sets a thread to be run during JVM termination
  - hook – created but not started thread
  - there can be several set hooks
    - they will start in some unspecified order
  - daemon threads run even during JVM termination
  - hooks are not executed if JVM was terminated using `halt()`

- **boolean removeShutdownHook(Thread hook)**
  - removes the set hook
  - return false if the given thread has not been set
Runtime

- **Process exec(String command)**
  - launches an external process
  - several variants (with different parameters)
  - may not always work correctly

- **the class Process**
  - represents an external process
  - methods
    - **void destroy()**
      - kills the process
    - **int exitValue()**
      - return value of the process
    - **int waitFor()**
      - waits until the process terminates
      - returns the return value
      - can be interrupted
java.lang.Math
java.lang.Math

- static fields and methods for basic mathematic constants and operations

- fields
  - PI, E

- methods
  - abs, ceil, floor, round, min, max,...
  - pow, sqrt,...
  - sin, cos, tan, asin, acos, atan,...
  - toDegrees, toRadians,...
  - ...