


Map Lecture Notes

<input checked="" type="checkbox"/> Archive	<input type="checkbox"/>
 Tags	

Lecture 1

- you can use `int value = Integer.parseInt(args[i])` to transform string to int
- only one class can be public per .java file
- if the access modifier is missing, the field is visible inside the package
- objects are created in the heap memory
- if the return type is not void, then each execution branch of that method must end with the statement `return`
- java doesn't allow operator overloading, class String has overloaded `+` `+=`
- there are no destructors → garbage collector
- a static method cannot use those fields(or call those methods) which are not static
- primitive type arguments are passed by value, the same for reference type
- strings are immutable
- covariant return type for overridden function
- no multiple inheritance, only multiple interfaces

Lecture 2

- static methods cannot be overridden
- java exceptions: Errors(external to the application), Checked Exceptions(subject to try-catch) and Runtime Exceptions(correspond to some bugs)
- finally always executes except: ctrl c and out of memory error

Lecture 3

- for generic classes you can't use primitive types for type variables(T)

- autoboxing: automatic conversion of a value of a primitive type to an object instance of a corresponding reference type when an object is expected and vice versa
- static methods cannot use the type variables
- generic method can have different type variables, can be defined in a non-generic class
- avoid using raw types
- generic arrays: cannot be created using new, error at compile time
- if T has constraints then through T we can call any method from the class and interfaces specified as bounds
- Wildcards ?
 - upper bound: extends class or interface
 - lower bound: super C ; any superclass of C

Lecture 4

- IO
 - bytes:
 - InputStream: abstract class for reading bytes from stream
 - OutputStream: abstract class for writing bytes into stream
 - chars:
 - Reader: abstract class for reading chars from stream
 - Writer: abstract class for writing chars into stream
 - conversion: InputStreamReader, OutputStreamWriter

Classes

Operations	Byte	Char
Files	FileInputStream, FileOutputStream	FileReader, FileWriter
Memory	ByteArrayInputStream, ByteArrayOutputStream	CharArrayReader CharArrayWriter
Buffered Operations	BufferedInputStream BufferedOutputStream	BufferedReader BufferedWriter
Format	PrintStream	PrintWriter
Conversion Byte ↔ Char	InputStreamReader (byte -> char) OutputStreamWriter (char -> byte)	

- random access:
 - RandomAccessFile: allows random access to a file, uses a cursor to denote the current position in the file, reading/writing move the cursor according to the number of bytes read/written
- Scanner
- Writing and reading data of primitive types: interfaces DataInput and DataOutput
- Class File contains the name of a file not its content, allow platform-independent operations on the files
- NIO
 - Channels:
 - all IO in NIO start with a Channel
 - both read and write(streams are one-way)
 - asynchronous
 - read to and write from a Buffer
 - FileChannel: reads data from files
 - DatagramChannel: UDP
 - SocketChannel: TCP

- ServerSocketChannel: listen for TCP
- can transfer data from channel to channel
- Buffers
 - block of memory into which you can write data and later read again
 - process:
 - write data into Buffer
 - buffer.flip() to switch mode
 - read data out of Buffer
 - buffer.clear() (clears all) or buffer.compact (clears only the data you have read) to make buffer ready for writing again
- Selectors: single thread to handle multiple channels
- Scattering reads: reads data from a single channels into multiple buffers
- Gathering writes: writes data from multiple buffers into a single channel
- AsynchronousFileChannel: via Future or CompletionHandler
- Try-with-resources

Lecture 6

- Java Serialization
 - serialization \Rightarrow convert object to byte stream and save into external source
 - classes need to implement interface Serializable(doesnt contain any methods)
 - readObject()/writeObject(Object)
 - if some values are wished to not be saved use keyword transient
- Functional programming
 - anonymous inner classes: provide a way to implement classes that may occur only once in an application
 - functional interfaces: one method
 - functional interfaces + anonymous inner classes are a common pattern
 - lambda expressions
 - created without belonging to any class

- can be passed around as if they were objects and executed on demand
- runnable lambda/comparator lambda
- Java Functional Streams
 - stream is a conceptually fixed data structure which elements are computed on demand
 - Stream processing phases
 - configuration: intermediate operations
 - filters:
 - filter(Predicate)
 - distinct: return a stream of unique elements
 - limit(n): returns a stream no longer than n
 - skip(n): return a stream where the first n elems are discarded
 - mappings
 - map
 - can form a pipeline
 - return a stream
 - are lazy: don't perform any processing, they only configure the stream, will be performed when the processing methods are invoked
 - processing: terminal operations
 - produce a result from a pipeline such as a List, Integer, void
 - collect
 - Collector.toList()
 - Collector.averagingInt(predicate)
 - Collectors.joining("delimiter", "prefix", "suffix")
 - Collectors.toMap(key, value, [(value1, value1) → predicate]), optional merge function to avoid duplicates
 - Stream.min()/Stream.max()
 - use get() to obtain the value
 - take a Comparator as parameter

- `Comparator.comparing(lamda exprssion)` i.e `item→item.length`
 - `count()` return the number of elements
 - `reduce([""],(acc,item) → acc + "" + item)`
- Numerical Streams
 - `IntStream`, `DoubleStream`, `LongStream`
 - convert stream to specialized: `mapToInt`, `mapToDouble`, `mapToLong`
 - generate ranges: `range` and `rangeClosed`
- Infinite streams using `generate` and `iterate` ex:
`Stream.iterate(0,n→n+10).`
- Finding and matching; `anyMatch`, `allMatch`, `noneMatch`
- Reusable Streams: `Supplier<Stream<TYPE>> streamSupplier = () → Stream.of(..)..`

Lecture 6

- Concurrent Programming
 - Processes
 - has a self-containted execution enviroment
 - in general it has a complete, private set of basic run-time resources i.e. has its own memory space
 - are fully isolated
 - to communicate they use pipes and sockets
 - Threads
 - lighthweight procesess
 - creating one requires fewer resources than creating a new process
 - exist withing a process - every process has at least one
 - share process resources
 - are units that are scheduled by the system for executing on the processor
 - define a thread:
 - extend the thread class + override the default Thread method `run()`, call `start()` to run a thread
 - implement the `Runnable` interface

- A sequential program is one that, when executed, has only one single flow of control, a multi-threaded program is one that can have multiple, a thread is a single sequential flow of control
- Synchronization
 - Synchronized methods and statements
 - Java uses the monitor concept to achieve mutual exclusion and synchronization between threads
 - `wait()`, `notify()`, `notifyAll()` control the synchronization of threads
- Java Monitor Model
 - a monitor is a collection of code (called the critical section) associated with an object (called the lock)
 - at any time instant only one thread at most can have its execution point located in the critical section associated with the lock
 - any object can be the lock of a monitor
 - a thread enters the critical section of a monitor by invoking `e.m()` or executing a synchronized statement; before it can run, it must first own the lock and will need to wait; a thread owning the lock will release the lock automatically once it exits the critical section
 - a thread executing in a monitor may encounter a condition in which it cannot continue but still does not want to exit → call `e.wait()` to enter the waiting list of the monitor → will release the lock so other threads have the chance to get the lock
 - a thread changing the monitor state should call `e.notify()` or `e.notifyAll()` to have one or all the threads compete with other outside threads for getting the lock
 - static methods cannot be synchronized, it belongs to the monitor whose lock object is `C.class`
- Thread groups: every thread is a member of a thread group: provide mechanism for collecting multiple threads into a single object and manipulating those threads at once rather than individually

Lecture 7

- executor service
 - interface for asynchronous execution mechanism, delegate tasks
 - `newSingleThreadExecutor()`/`newFixedThreadPool(n)`/`newScheduledThreadPool(n)`

- ways to delegate
 - `execute(Runnable)` - no way to get the result
 - `submit(Runnable)`
 - `submit(Callable)` - can be obtained via the Future object returned
 - `invokeAny/invokeAll`
- `Runnable` - void, `Callable` - value of Type Future used to retrieve the actual result later
- to terminate:
 - `shutdown()`: no longer accepts task and shuts down after all tasks finished
 - `shutdownNow()`: skip all submitted and non-processed tasks and shut down
- `forkjoinpool`
 - implements the work-stealing strategy, i.e. every time a running thread has to wait for some result, the thread removes the current tasks from the work queue and executes some other task ready to run. This way the current thread is not blocked and can be used to execute other tasks. Once the result for the originally suspended task has been computed the task gets executed again and the method `join()` returns the result
 - `fork()` - start an asynchronous execution of the task, `join()` wait until the task is finished and get the result
- interface `blocking queue`
 - throws an exception: `add(e)`, `remove()`, `element()`
 - return null or false: `offer(e)`, `poll()`, `peek()`
 - blocks the current thread indefinitely until the operation can succeed: `put(e)`, `take()`
 - blocks for only a given maximum time limit before giving up: `offer(e, time, unit)`, `poll(time, unit)`
- concurrent collections: `concurrent HashMap`
- semaphore
 - a synchronization aid provided by `java.util.concurrent` package that allows only a certain amount of threads to access a critical section
 - the semaphore is initialised with a given number of permits; for each call to acquire a permit is taken by the calling thread; for each call

to release a permit is returned to the semaphore; at most n threads can acquire a permit at the same time, n is the initial number of permits

- `CountDownLatch`
 - a synchronization aid provided by `java.util.concurrent` package that allows one or more threads to wait until a set of operation being performed in others threads completes
 - is initialized with a given count; this count is decremented by calls to the `countDown()` method; threads waiting for this count to reach zero can call one of the `await()` methods (simple `await` or `await` with timeout which waits for count to reach zero or the specified waiting time elapses); calling `await` blocks the thread until count reaches zero
- `CyclicBarrier`
 - is a synchronization mechanism provided by `java.util.concurrent` that can synchronize threads progressing through some algorithms; its cyclic because it can be reused after all the threads have reached it by calling the `reset()` method
 - it is a barrier that all threads must wait at, until all threads reach it, before any of the threads can continue; once N threads are waiting at the Cyclic Barrier all the threads are released and can continue running;
 - you can specify a `Runnable` task to be executed once the barrier is tripped i.e once the specified number of parties has reached the barrier
- `lock`
 - like synchronized blocks but more sophisticated
 - Reentrant Lock
 - `ReadWriteLock`
 - multiple threads can read but only one can write at a time
 - `ReadLock`: use if no threads write or requested to write
 - `WriteLock`: no threads are reading or writing
- atomic variables
 - atomic Integer, atomic Boolean, atomic methods >> synchronized

Static vs Non-static methods

Points	Static	Non-Static
Def	method that belongs to a class but doesn't belong to an instance of a class, it is without an instance of that class	method that each instance has, every method without static keyword is non-static, it needs an instance to call it
Overriding	cannot be overridden because they use compile-time or early binding	can be overridden because they use runtime/dynamic/late binding
Accessing members and methods	can only access static members and methods	can access both static and non static methods and members
Usage of Type variables	No	Yes
Memory allocation	Only once	Every time is called

Checked vs Unchecked exceptions

Points	Checked	Unchecked
Examples	IO/Compile Time exceptions	Runtime or Null Pointer exceptions
Def	checked at compile time; if some code throws a checked exception, it must handle it(try and catch) or specify the exception using the throws keyword and the called of the method must handle the exception	These are exceptions that are not checked at compile time; they are runtime exceptions because they are not required to be caught or declared in a throws clause, they are usually programming errors; include all subclasses of RuntimeException/Error classes

Streams vs Collections

Points	Streams	Collections
Def	a stream is a conceptually fixed data structure in which elements are computed on demand	in-memory data structure, which holds the vales that the data structure currently has-every element in the collection has to be computed before it can be added to the collection

Pipelining	many stream operations return a stream themselves → chaining, laziness and short-circuiting	No
Internal iteration	stream operations do the iteration behind the scenes for you	No, iterated explicitly

Interface vs Abstract Class

Points	Abstract Class	Interface
Accesss Modifiers	public,protected,private methods	only public methods
Fields	Yes	Only static and final
Constructors	Yes	No
Methods	can have no abstract methods	can have 0 methods
Instance obj	NO	NO
Multiple inheritance	No	yes

Overloading vs Overriding

Points	Overloading	Overriding
Polymorphism	Static/early/compile binding	dynamic/late/runtime binding
Features	Same function, same return type but different signature	Same function but in classes parent-child using @Override keyword, the return type may be a subtype of the return type in the overridden method
Private and final methods	Can be overloaded	Cannot be overridden
Static Methods	can be overloaded	Cannot be overridden
Inheritance	May or may not require	always requires

Reference Type vs Primitive Type

Points	Primitive	Reference
Storage	store the actual value directly	store memory address to objects
Memory efficiency	high efficiency due to direct storage	les memory-efficient due to storing references
access-speed	faster	slower
usage	simple values	complex objects, arrays, custom classes
sharing data	independent	can share data among different parts

Wildcard

Upperbounded- read only ? extends

lowerbounded- write only ? super

unbounded- ?

Threads vs Processes

Points	Threads	Processes
	means a segment of a process	means any program in execution
	less time to terminate and create	more time to terminate and create
	less time for context switching	more time for context switching
	share memory	isolated
	if one blocked, then all are blocked	blocking one doesnt affect other