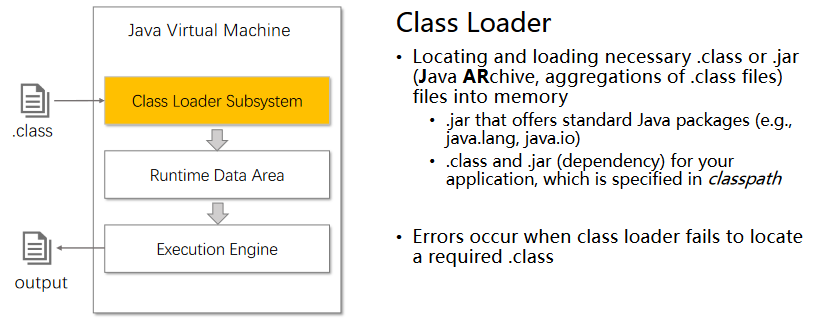
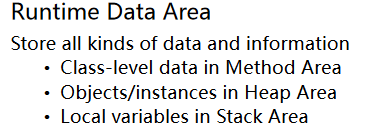
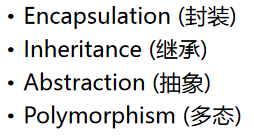
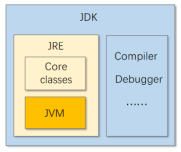
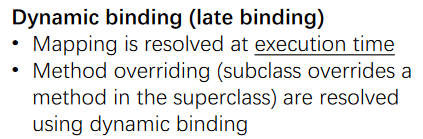
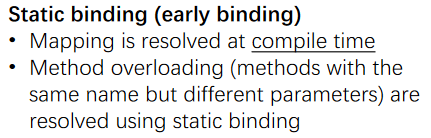
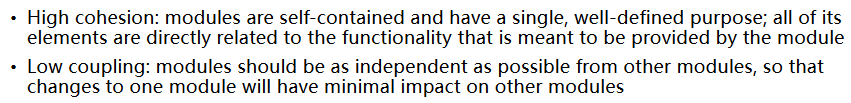
**Lecture1**

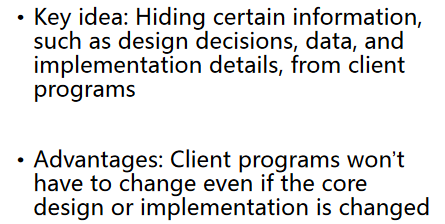












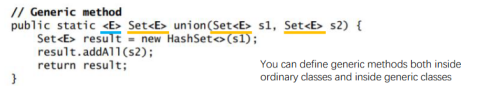
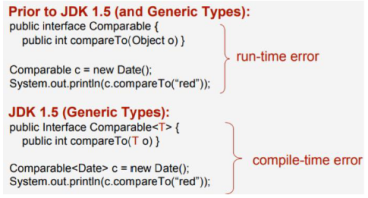
**Lecture2 Generics, Collections**

**Generics**

E non-primitive type

why? type erasure; primitive is stored in stack

Generic classes, interfaces, methods



<T extends BoundingType1&type2>

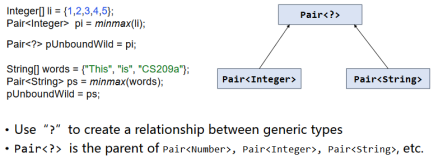
T could be any subtype of 1,2

Inheritance

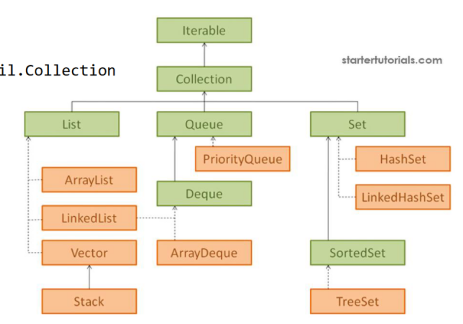
Number←Integer √

Pair<Number>←Pair<Integer> ×

Wildcards通配符

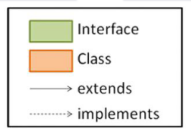


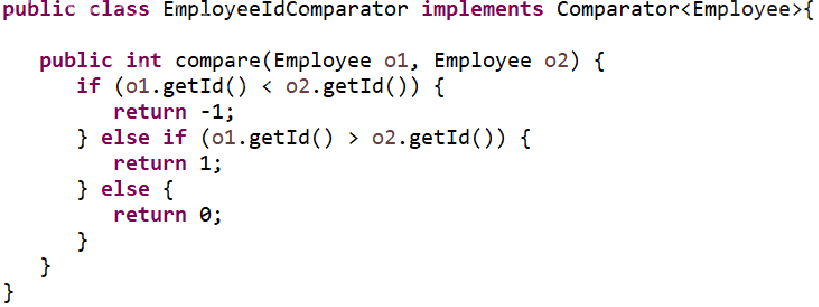
**Collections**



Java2 Cheating Paper

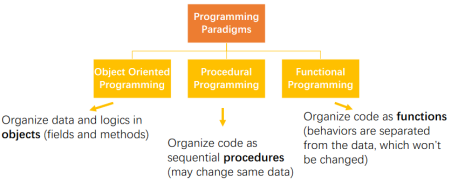
@Jiko





**Lecture 3 Lambda**

**Functional Programming**



Pure function

Always produce the same output for the same input. *Copy, create rather than modify.*

No “while” or “for” loop in functional programming. *Use recursion rather than iteration.*

Pros:

Easy to debug, test, and parallelize.

Complexity is dramatically reduced

**Lambda Expression**

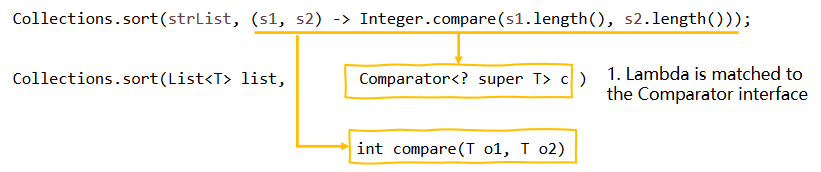
Syntax

(param1, param2) -> {expressions or statements, and return}

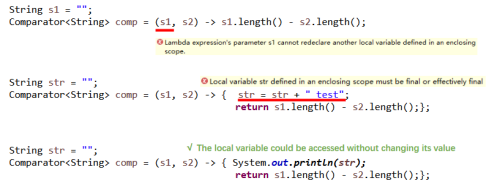
*(param1, param2) -> {return param1 > param2;}*

*(param1, param2) -> param1 > param2;*

Functional interface is an interface with a single abstract method.

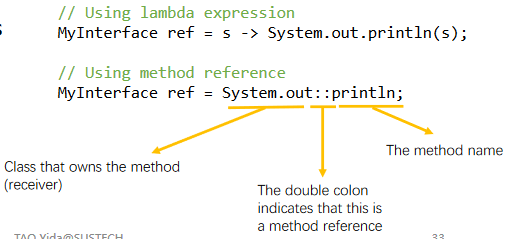


Type inference



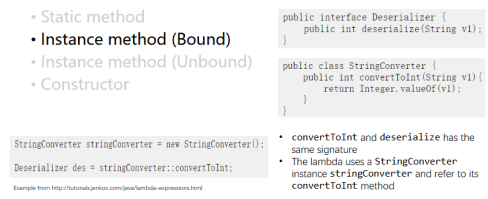
Use cases

Method references

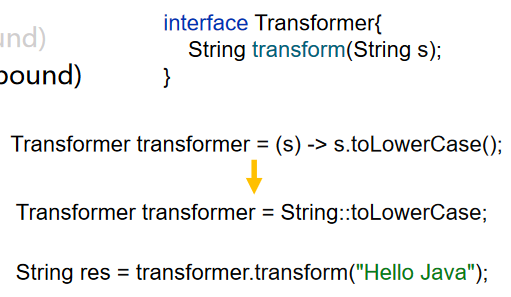


Static method”ClassName::staticMethod”

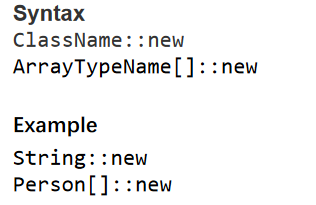
Instance method(Bound)”InstanceName::instanceMethod”



Instance method(Unbound)



Constructor.



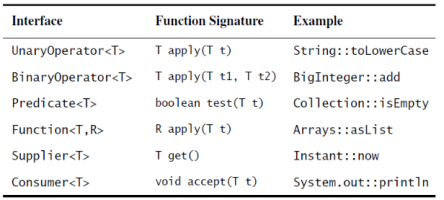
Java functional interfaces

Consumer<T> 是一个消费者接口，表示接受一个参数（类型为T），并对该参数执行某些操作，但没有返回值。

Supplier<T> 是一个供应商接口，表示提供一个值（类型为T），但不接受任何参数。

Predicate<T> 是一个谓词接口，表示对输入参数进行某种判断，返回一个布尔值。

Function<T, R> 是一个函数接口，表示接受一个参数（类型为T），并返回一个结果（类型为R）。



**Lecture 4 Stream**

**Java 8 Stream**

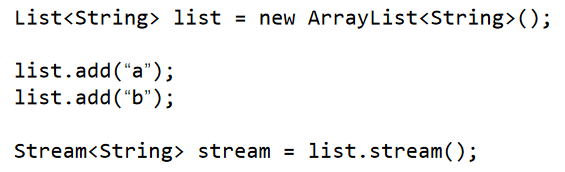
Used to process collections of objects

Data stream is obtained from a source

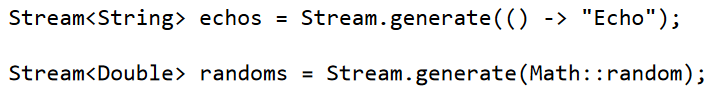
Data stream is processed through chained intermediate operation

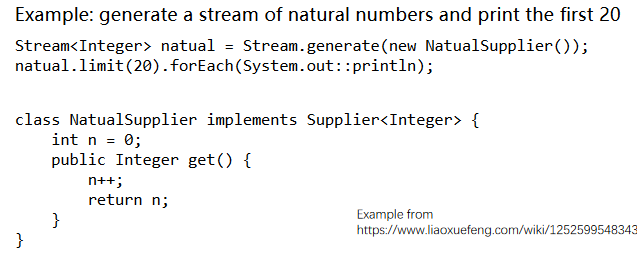
Getting the result from a terminal operation

*Create*

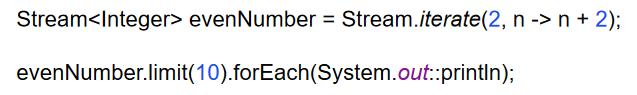


*Create2*

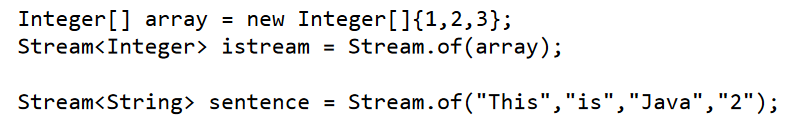




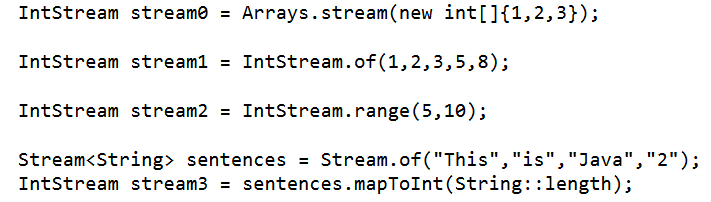
*Create3*



*Create4*



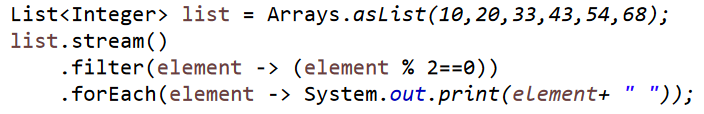
*Primitive generation*



*Intermediate Operations*

[Lazy evaluation] All intermediate operations do not get executed until a terminal operation is invoked

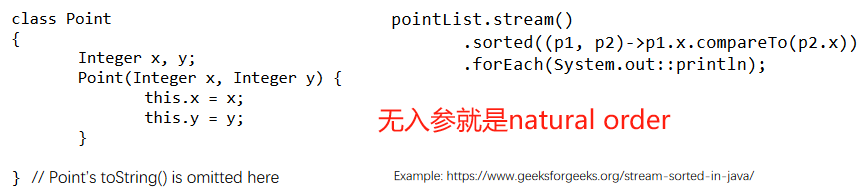
*filter()* match



*map()*



*sorted()*



*Distinct(),peek(),limit(),skip()*

*Terminal Operations*

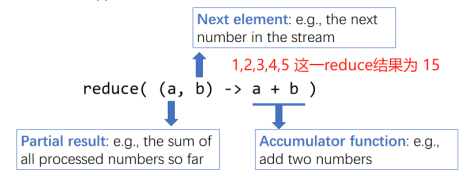
A terminal operation returns a non-stream

type of result. [Eager execution] Terminal operations are

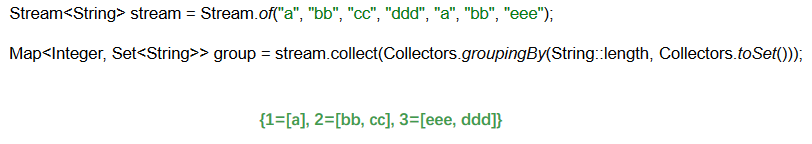
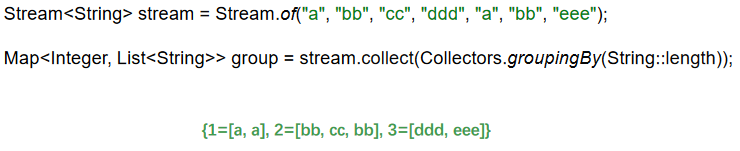
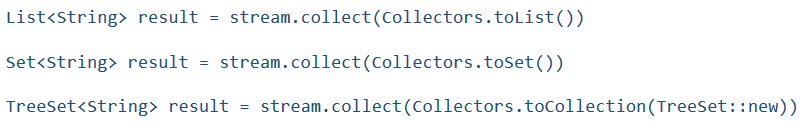
early executed

*Reduction*

min(),max(),average(),sum(),reduce()



*Collecting Results*



Lazy Evaluation:

延迟计算，节省资源（优化中间操作），并行处理，灵活性（可以在不同的流操作之间插入新的操作）

**Optional<T> class**

a type-level solution for representing optional values instead of null references

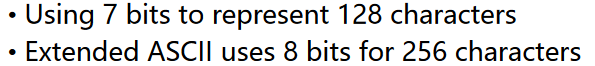
**Lecture 5 I/0**

**I/O**

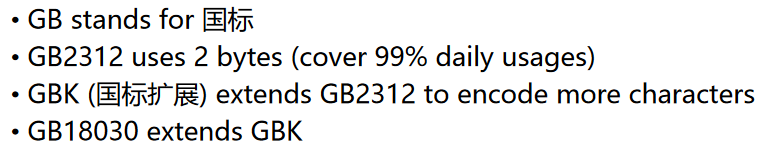
Input Stream and Output Stream are Byte Stream, Reader and Writer are Character stream

**i18n & Character Encoding**

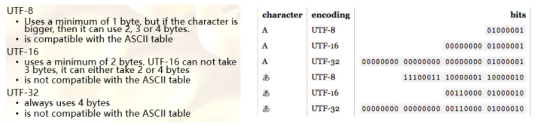
*ASCII*

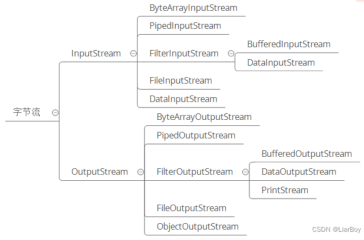


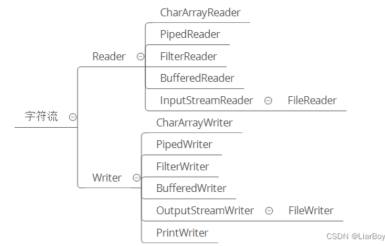
*GB*



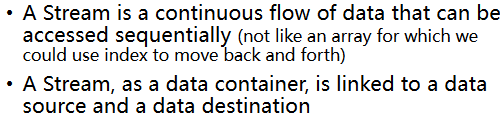
*Unicode*

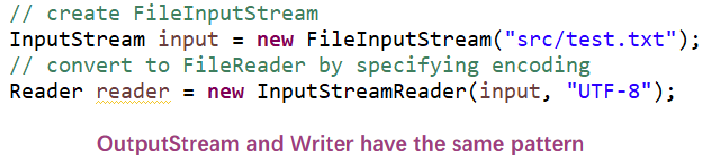






**Byte Streams & Character Streams**





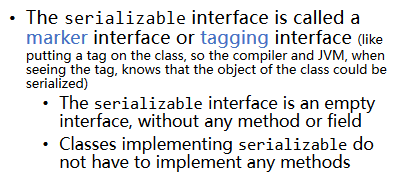
**Combining Stream Filters**

**Reading/Writing Text Input/Output**

**I/O from Command Line**

**Lecture 6 Exception**

**Persistence and Serialization**

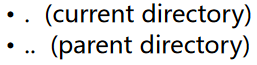


ObjectOutputStream.writeObject(myObject)

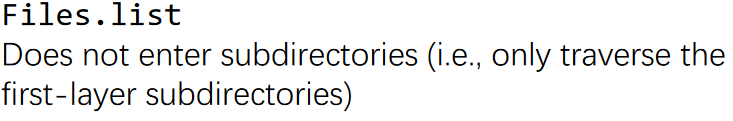
ObjectInputStream.readObject()

**Working with Files**

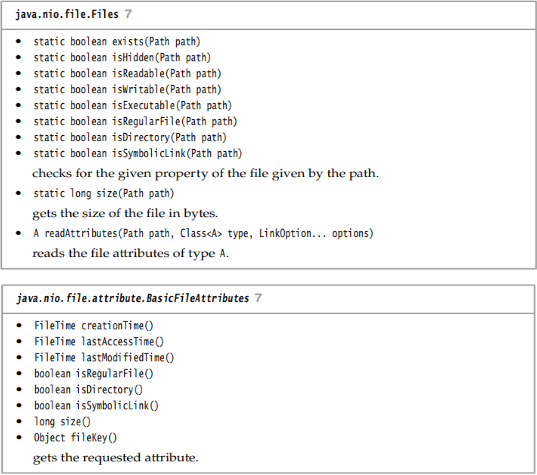
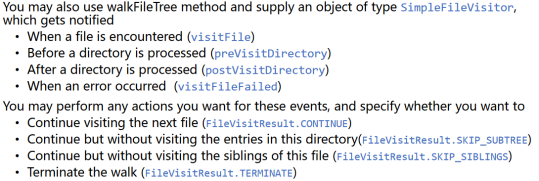
Path







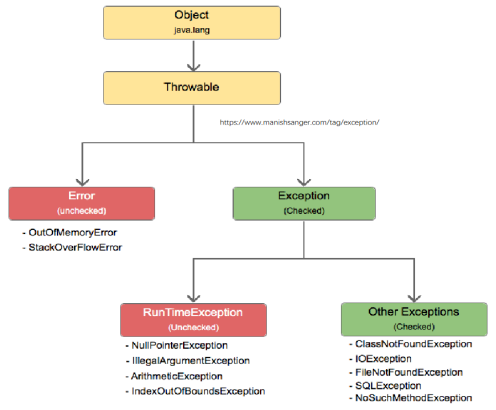


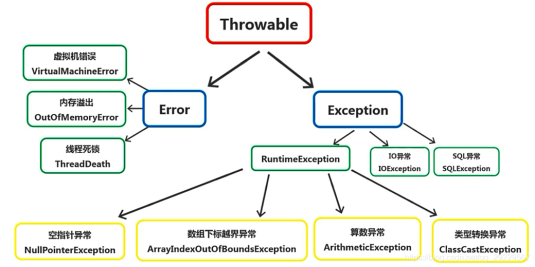


**Exception Handling**









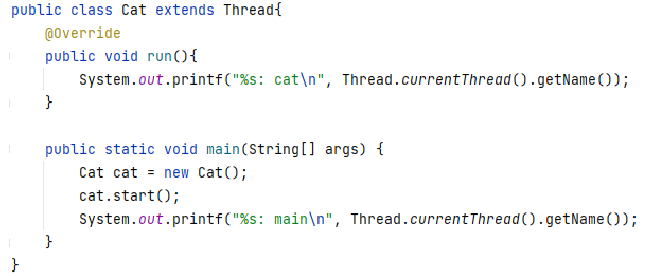
**Lecture 7 Concurrency**

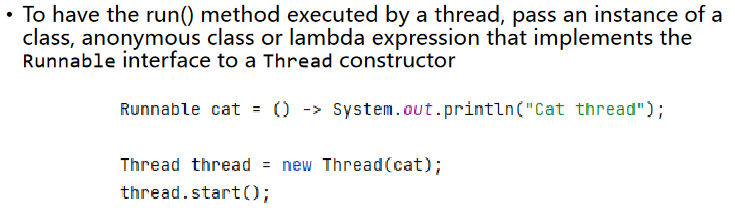
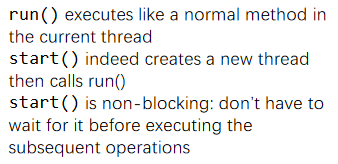
**Overview**

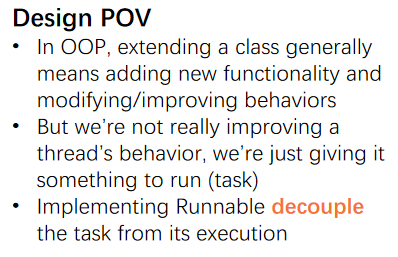
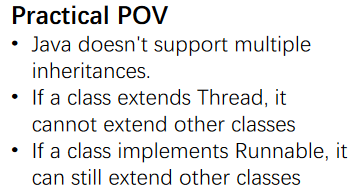
Processes进程，Threads线程 一个进程有多个线程

**Creating & Starting Threads**

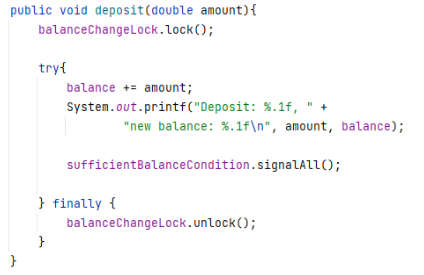
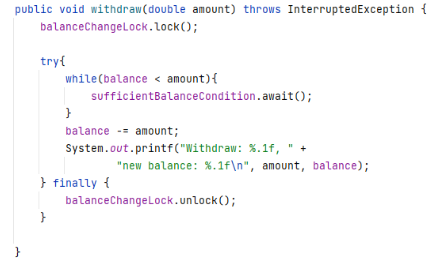
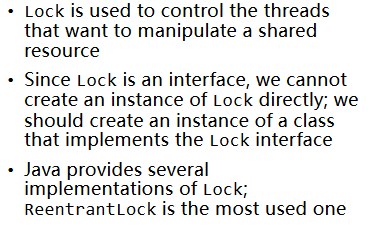
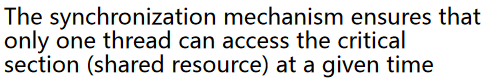
Extends Thread class or Implement Runnable interface(better)

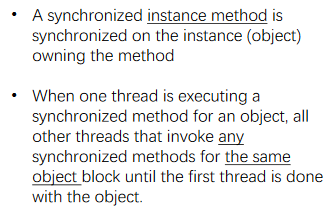


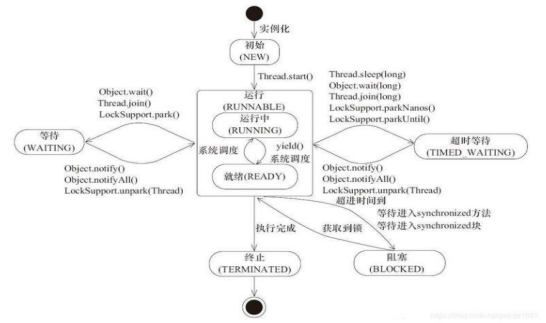




**Synchronization**







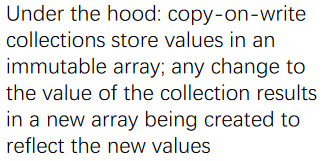
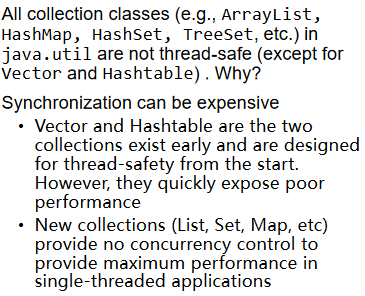
Volatile

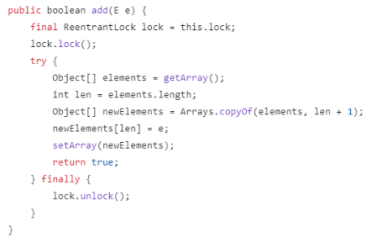
Visibility: 每次访问该变量时都会强制刷新缓存

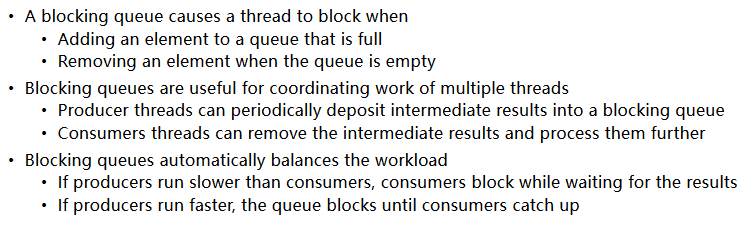
Ordering: 禁止指令重排序，确保有序性

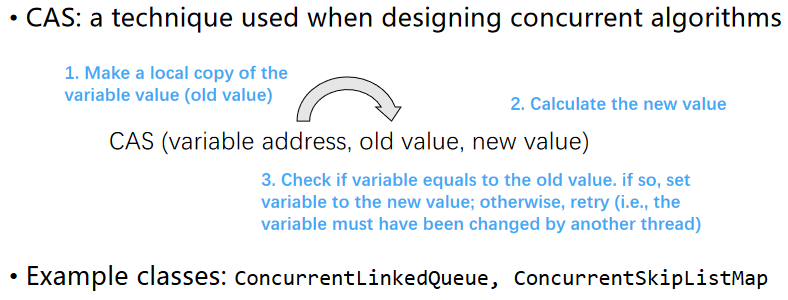
不保证原子性，如i++

**Thread-safe Collections**

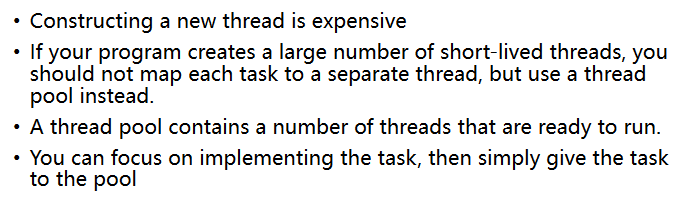


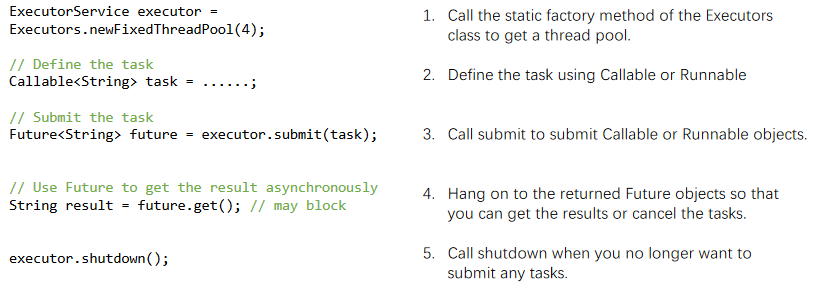
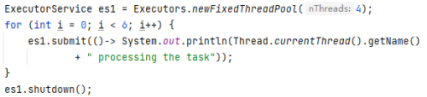
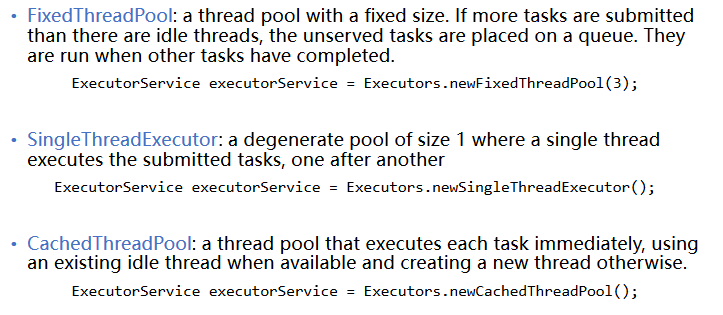
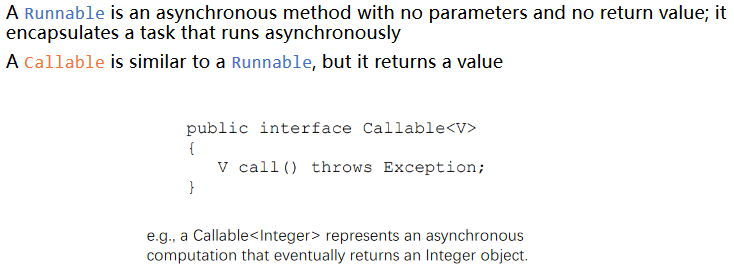


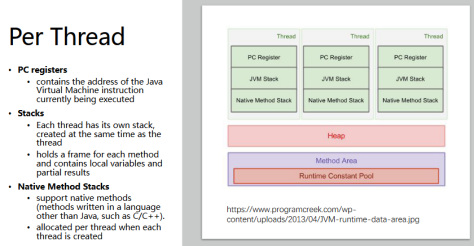
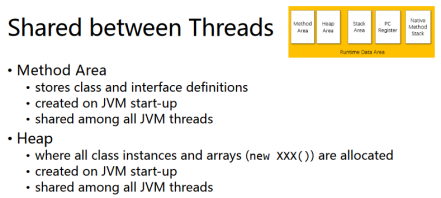




**Tasks and Thread Pools**

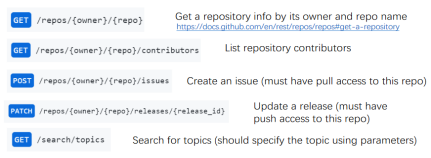


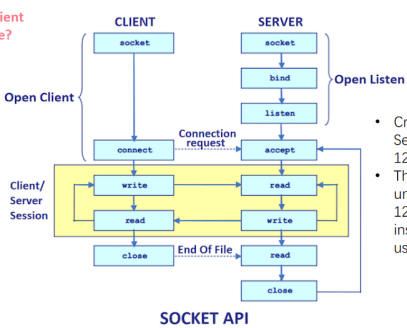


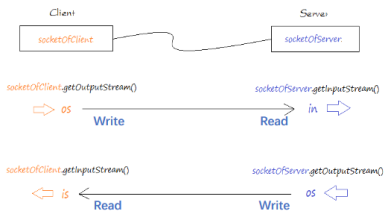


**Lecture 8 Socket**

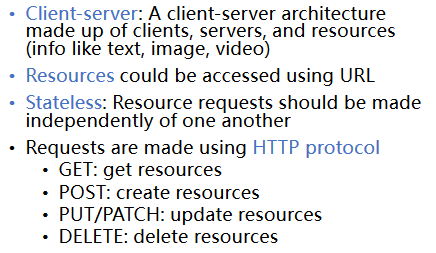
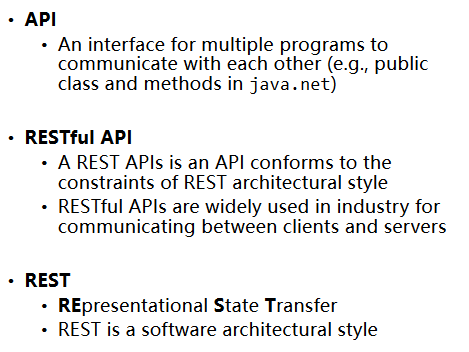
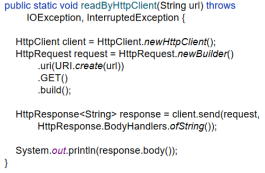
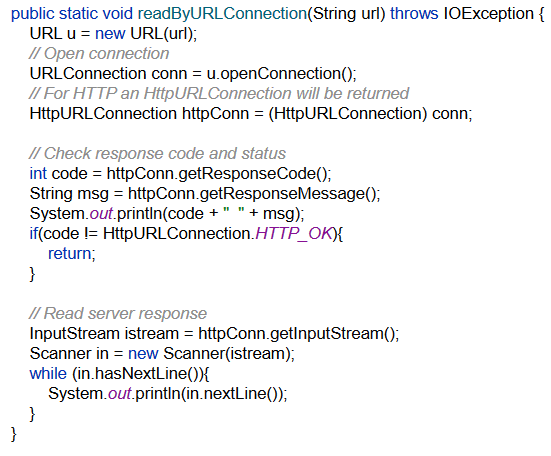
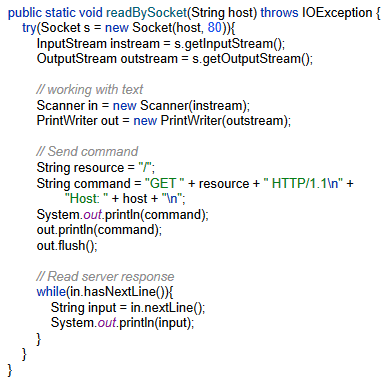
**Socket Programming**





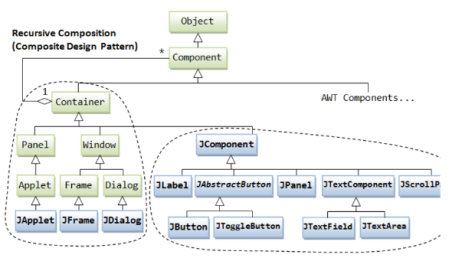


**Getting Web Data**

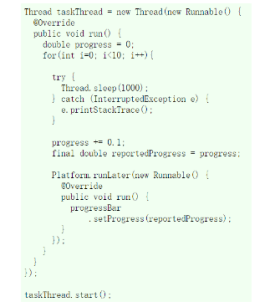
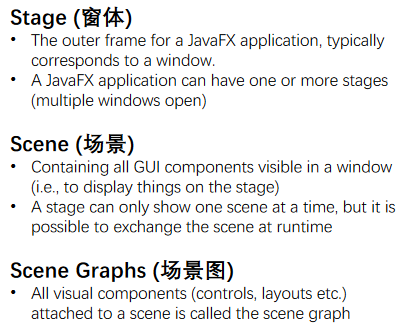


**Lecture 9 JavaFX GUI**

**Swing**

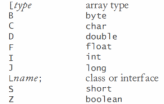
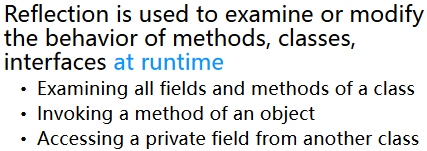


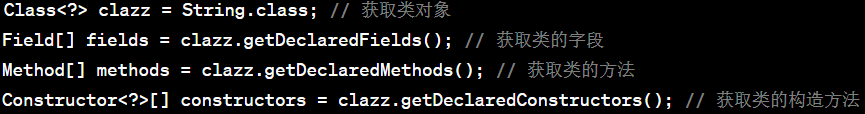
**JavaFX**

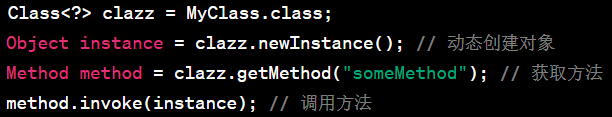


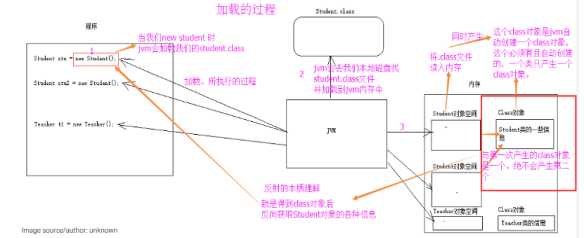
**Lecture 10 Reflection**

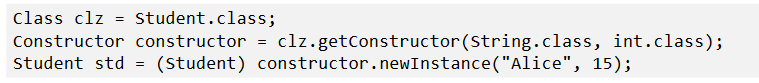
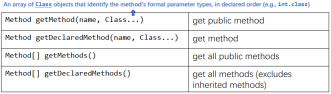
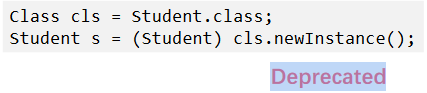
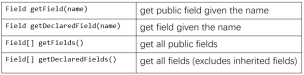
**Reflection**

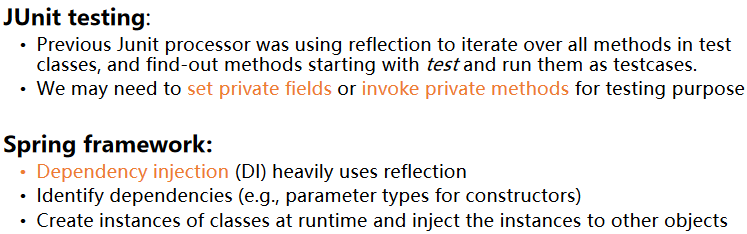


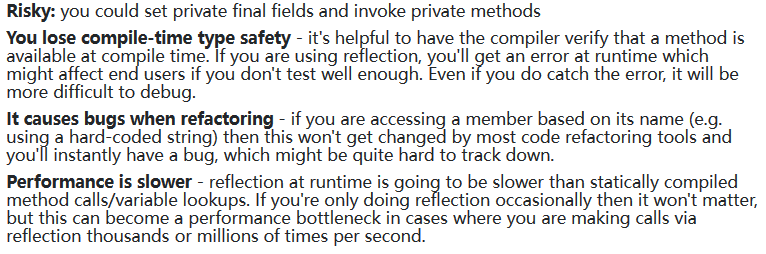




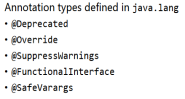
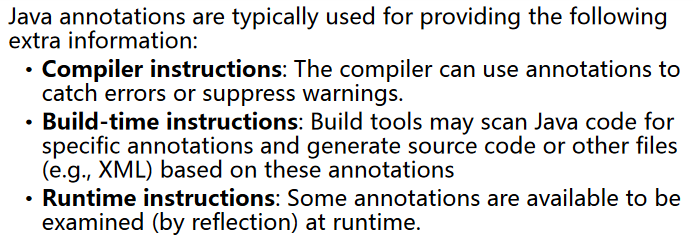


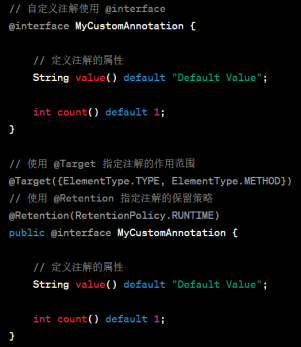
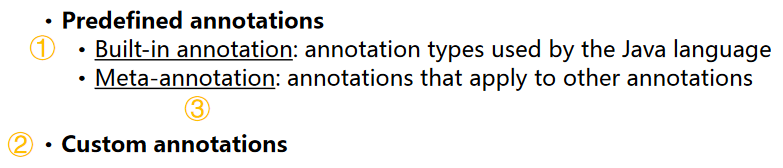
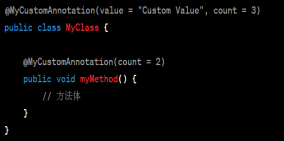


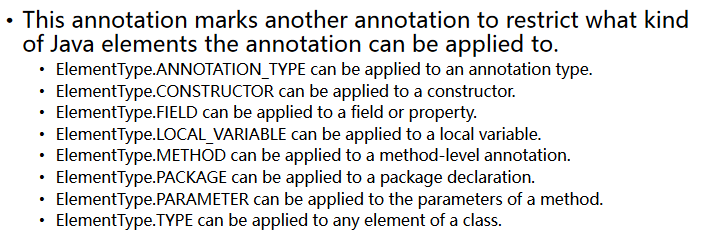


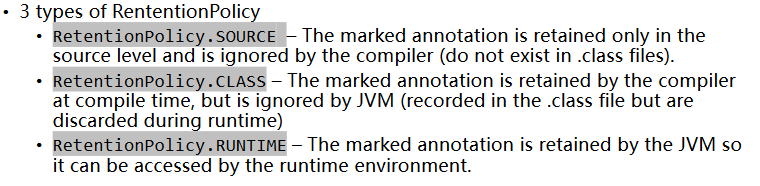


**Annotation**



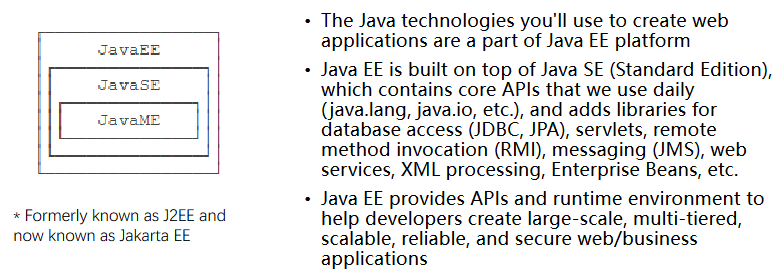


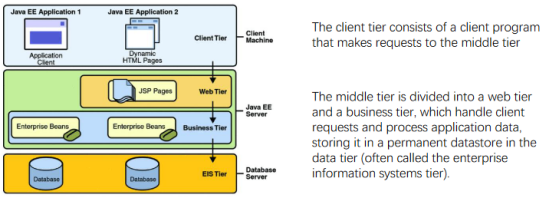




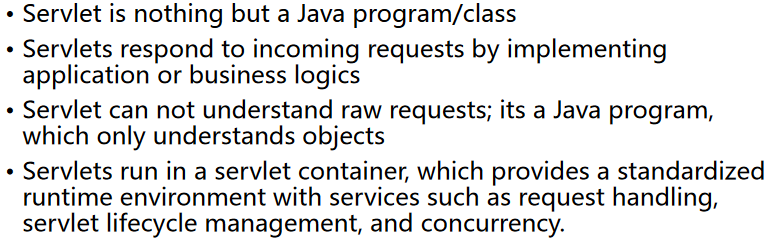
**Lecture 11 Java EE**

**Java EE**

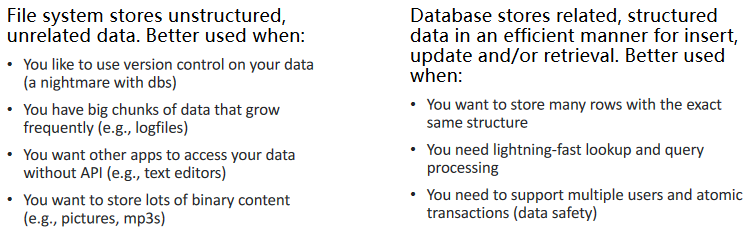




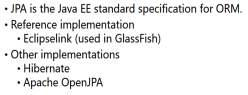
**Servlet & Containers**

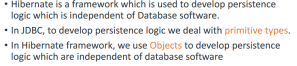


**JDBC & JPA**



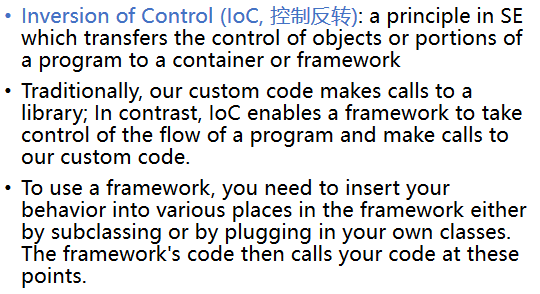
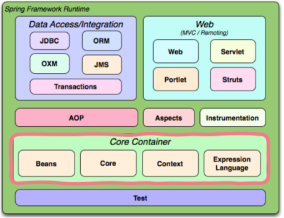


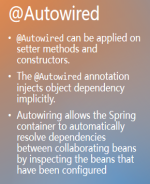
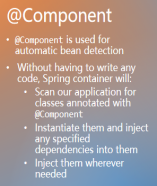
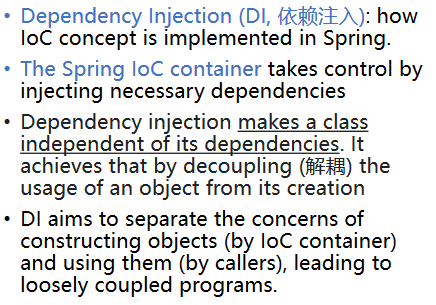


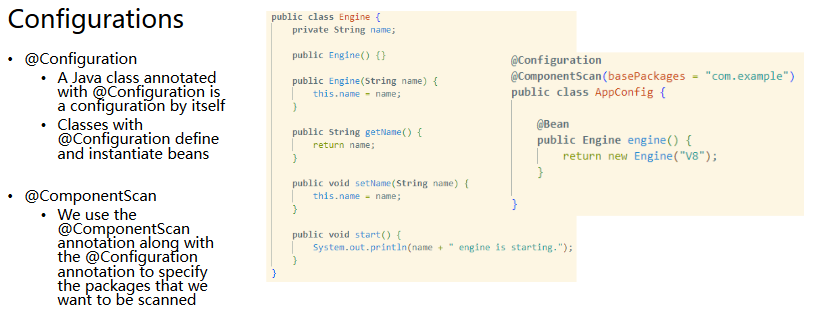


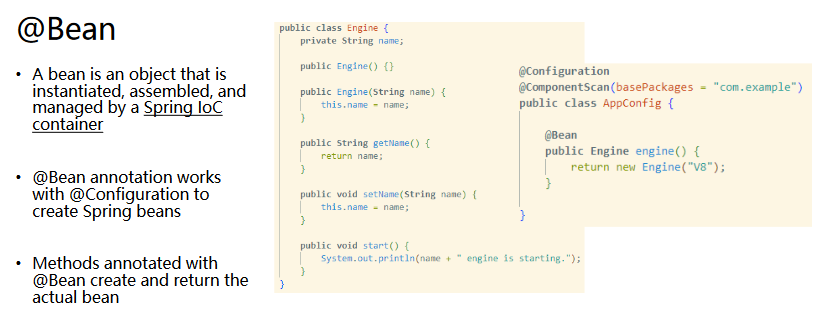
**Lecture 12 Spring**

**Spring Framework**

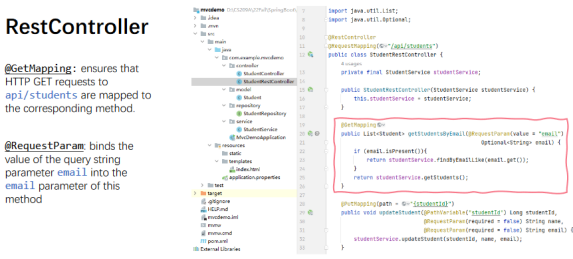
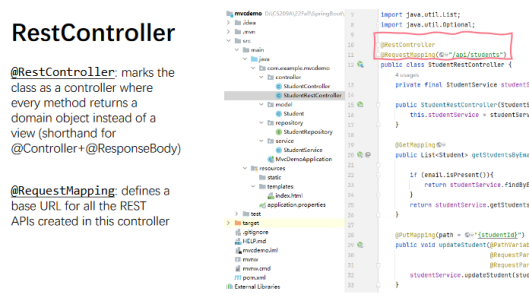
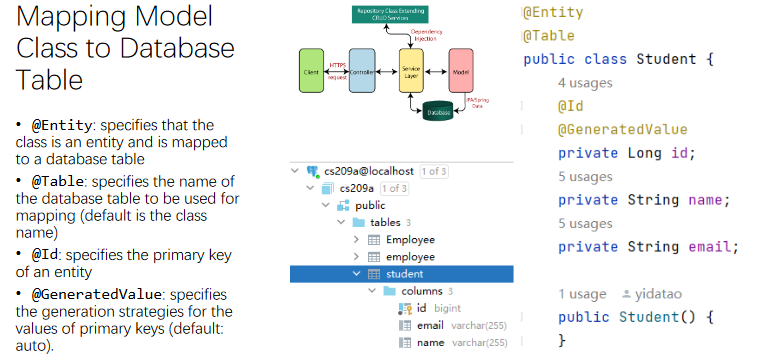
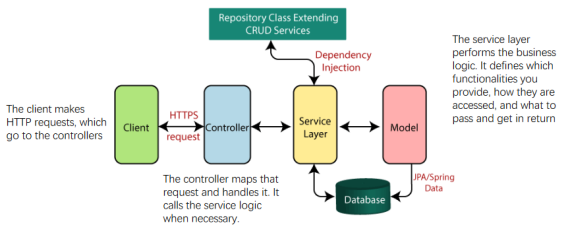






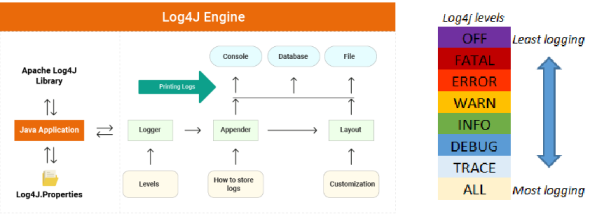


**Spring Boot**

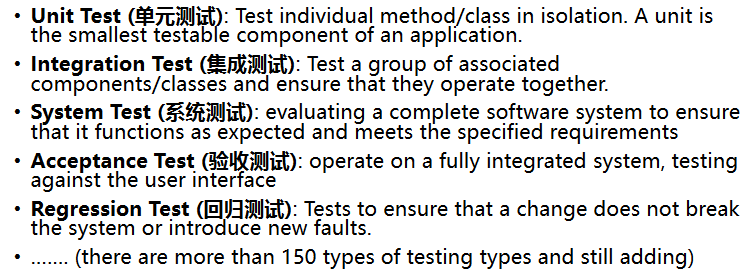


**Lecture 13**

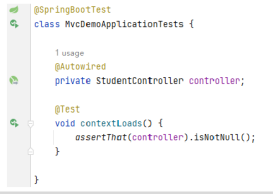
**Logging**



**Testing**

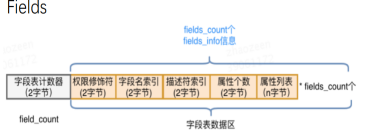
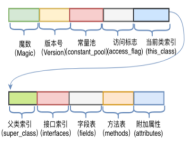


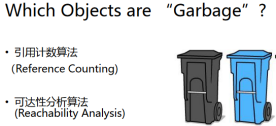




**Lecture 14 JVM**

**.class structure**





**JVM**

