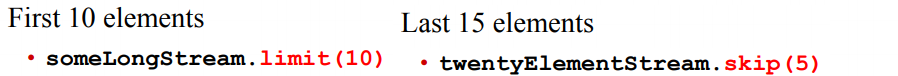
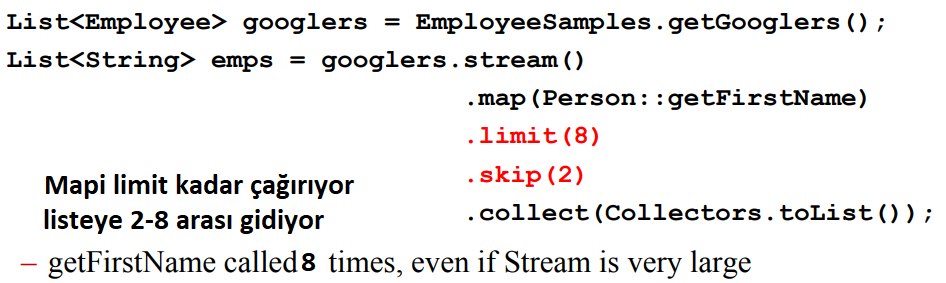
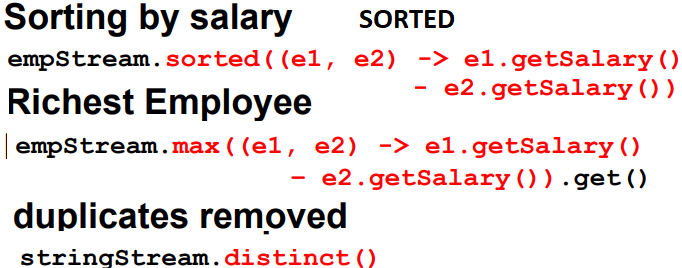


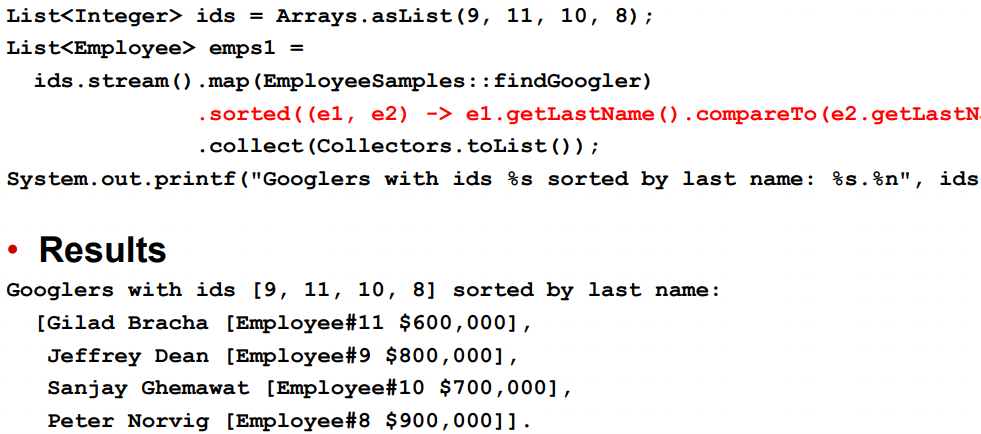
Map(findGoogler).filter(checkFornull).







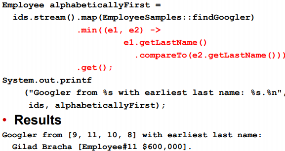
If the Stream elements implement Comparable, you may omit the lambda and just use str.sorted().



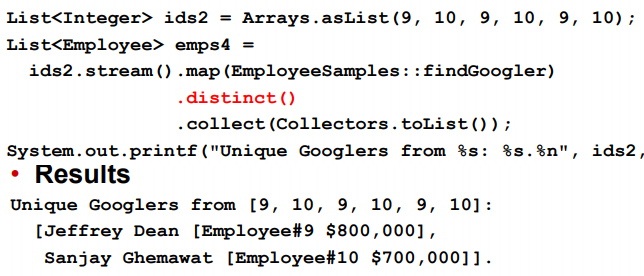
Sorted(...).limit(2) does not reduce

number of times that sorted is called.

**Min,Max:** Unlike with sorted, you must provide a lambda, regardless of whether or not the Stream elements implement Comparable. Using min and max is faster than sorting in forward or reverse order, findFirst + min and max->O(n),sorted->O(nlog n).



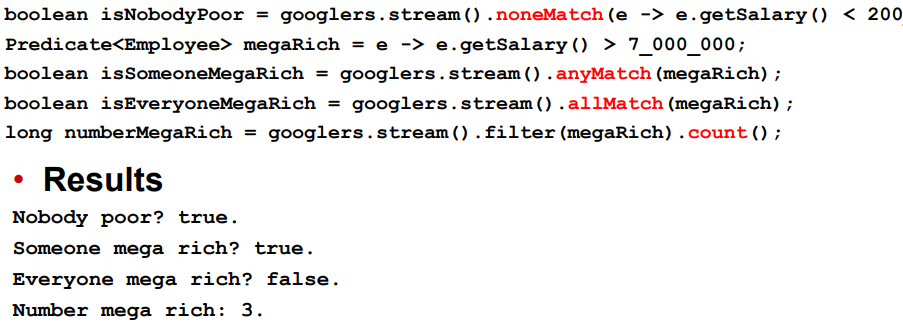
**Distinct** example



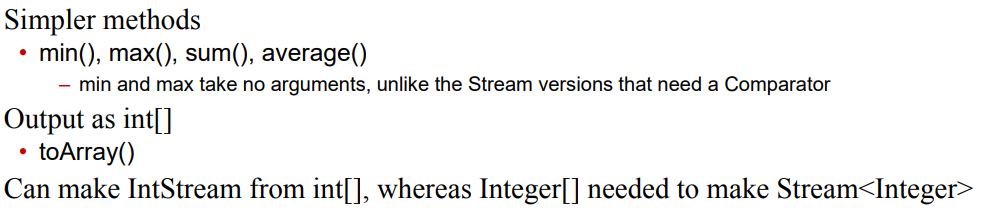
**Matches**, allMatch, anyMatch, noneMatch. Take predicate, ret. bool

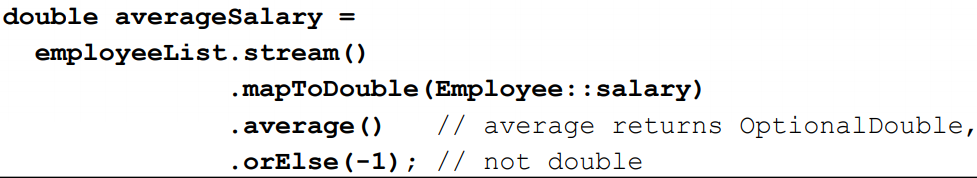


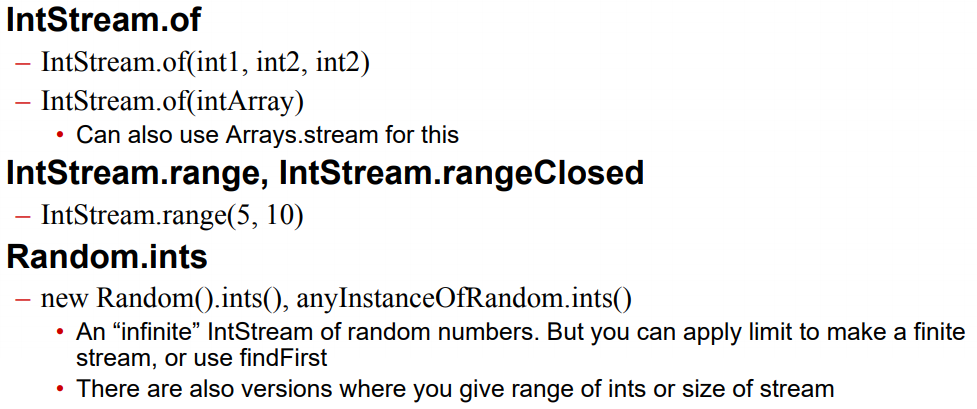
If the first element fails the Predicate, **allMatch** would immediately return false and skip checking other elem.

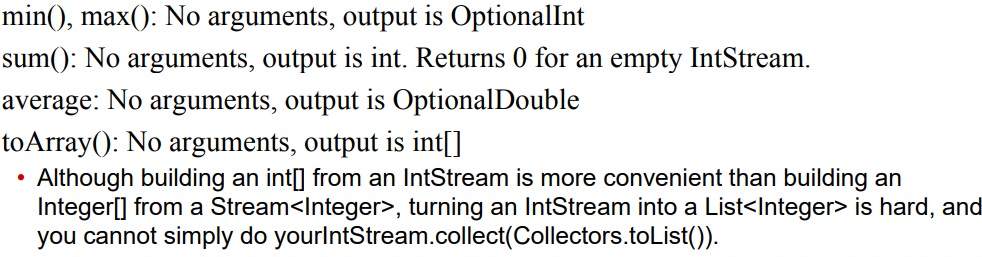


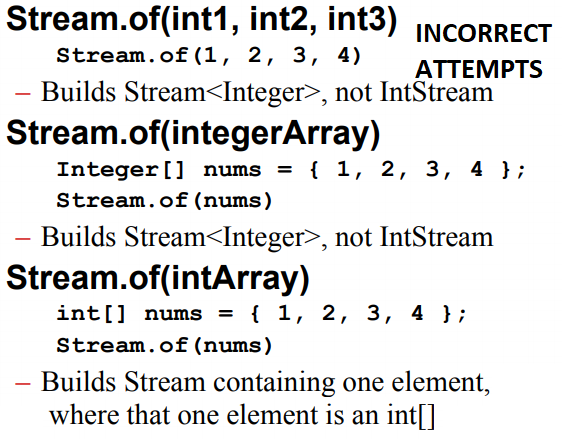
**IntStream**:



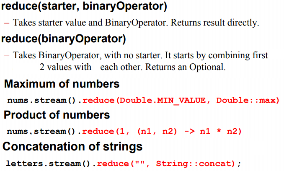


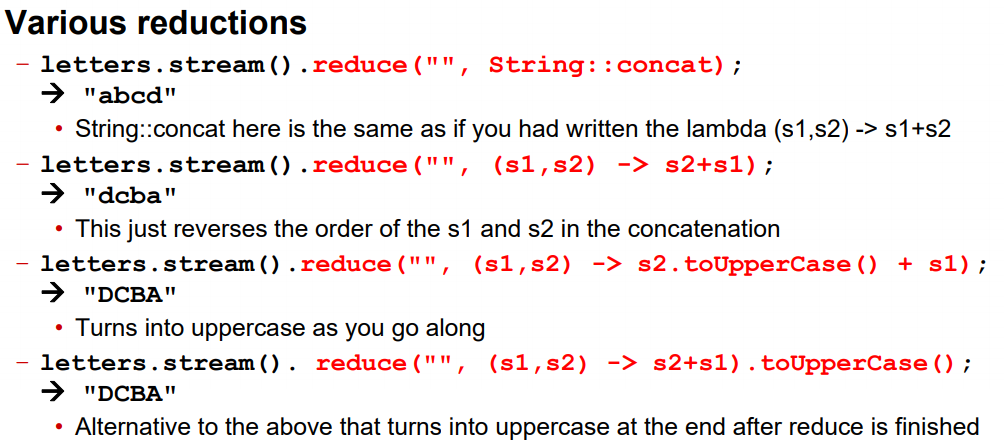


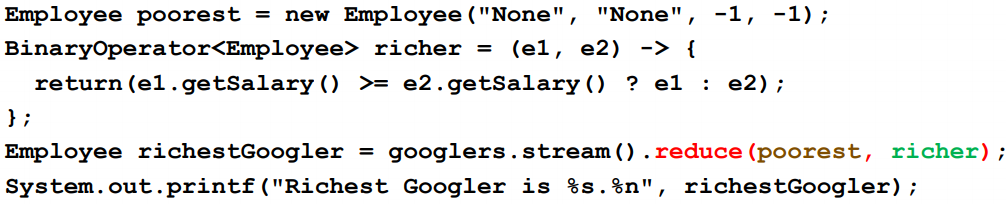


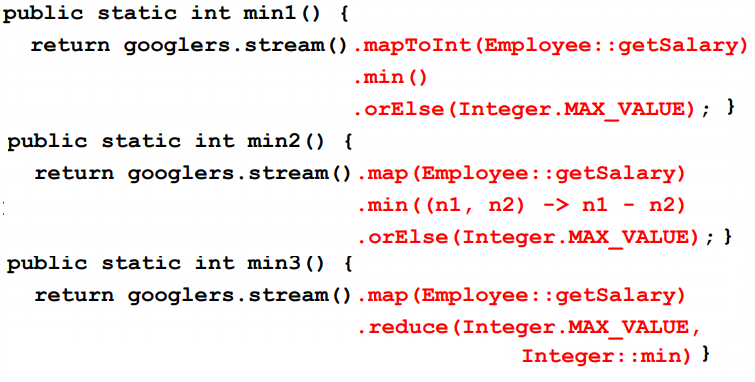


**Reduction** op. take a Stream<T>, and combine or compare the entries to produce a single value of type T

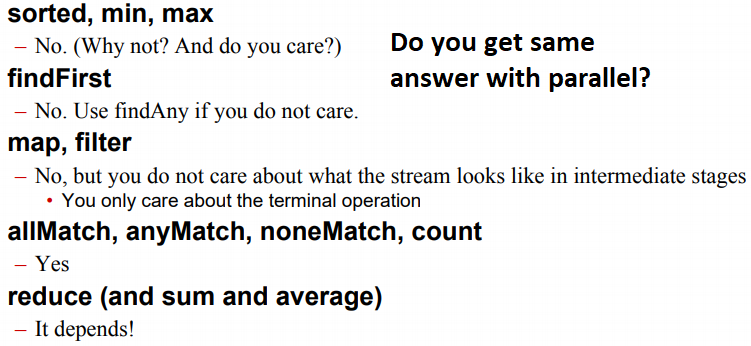


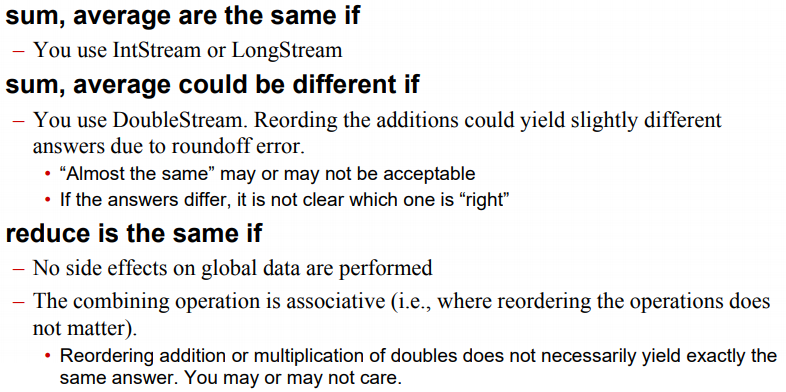


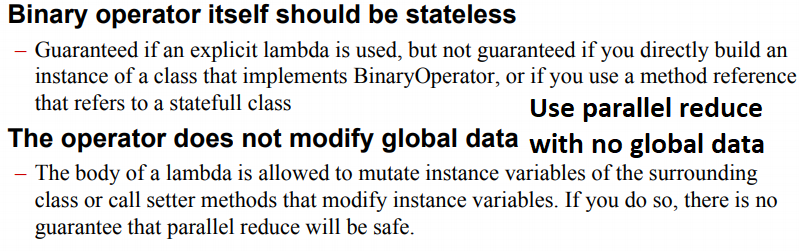


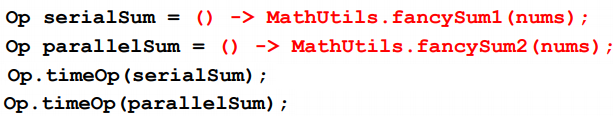


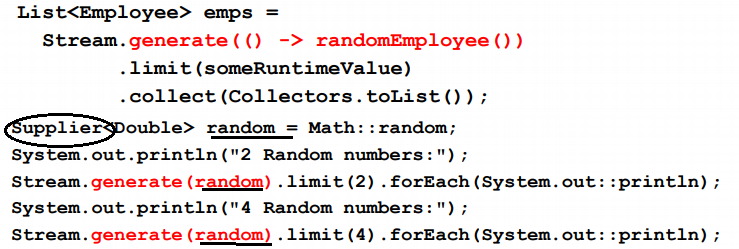
Parallel Streams: No benefit on single core, fork/join. Easier than threads.

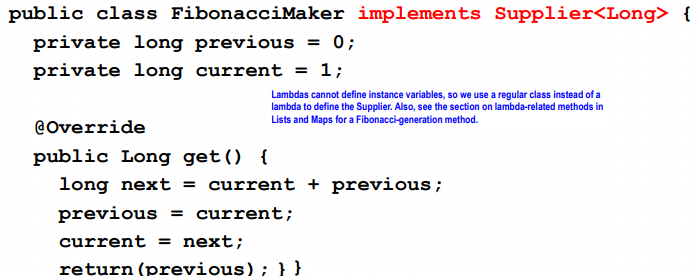


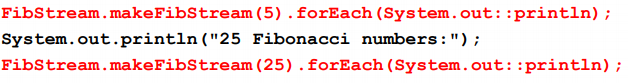


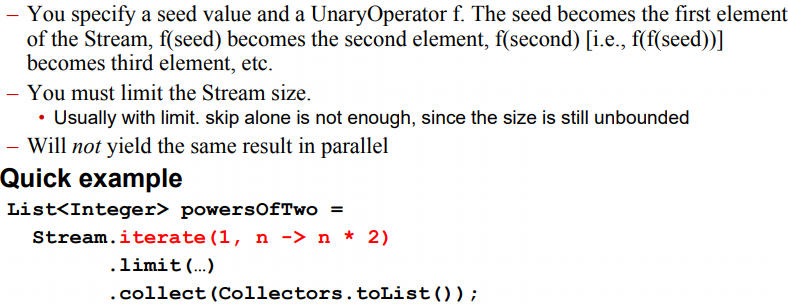


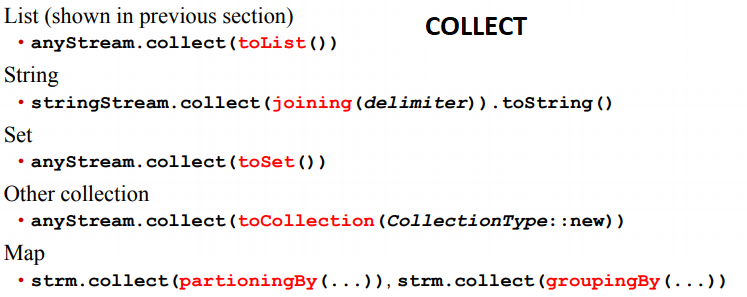


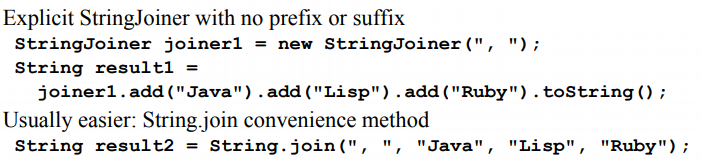


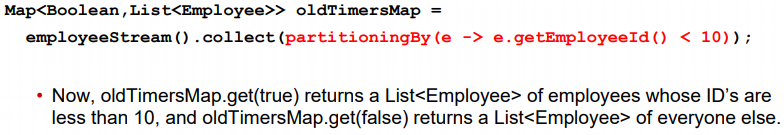


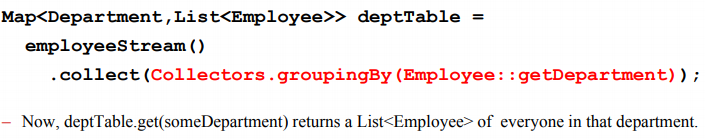


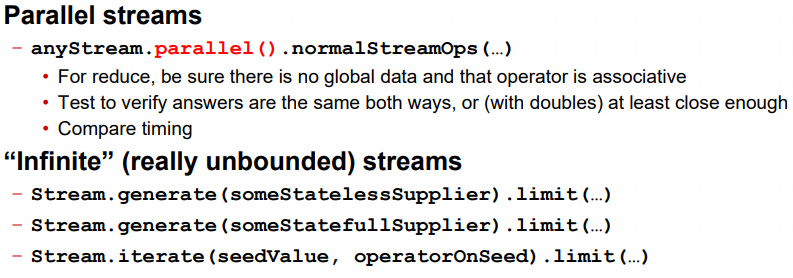








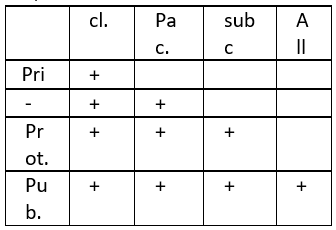




**StrBuffer** Mut, sync (mult.thr), **StrBuilder** mut, not sync

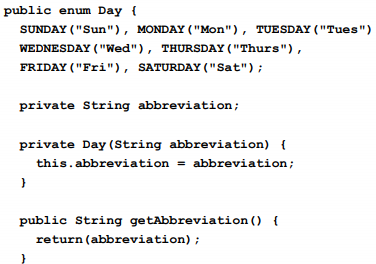
**Array**, each dimension special obj. Size of arr cant modified. Intarr init as 0, others null,. Circle[] ci = new C[size] foreach (c:ci)c=newC() NO. Dont init **Autoboxing**, double->Double **Inheritance** Pub cl Child ext Par{}, child access **all non private** of Par, access private with get/set. If no extends-> extends Object. Construc: first line is **super()**, by default. **Attempt1:** sumAr(Obj[] ob){ for(..)sum+= ob.ar()} Fails. Ob don’t have getArea in its view. **Attempt2:** Create Shape, getArea()=-1. Cir.ext Sha{..} **Attempt3: Abstract Class**

**Attempt4: Interface** We don’t modify a data member, use int. If a cl. imp. Sh but not getAr.(), that class is abstract. Cl A imp IA{...}. IA ia = new A(); IA,f()<-A,g(). Same issue. Cant say ia.g(). g() does not exist in IA.



**Final** -> it cnt be changed after init.

fin Cl -> cant ext. , fin f()->can’t override. **Volatile**: don’t cache, every th. Will get it from memory, will see update. Don’t use for race conditions. **Native**: C/C++ imp. **@Override**-> cannot narrow visibl. **Enums** enum E => cl E ext java.langEnu{} Enum doesn’t extend. Can have pr./pub. members. **Constructors are private.**

****

When day is referred, **all constr**. will be called. Day d1 = Day.MONDAY, d1.get(). **Generic**. List<Emp.> workers=..; Wor.add(new Emp()). Type check at comp. Time. After comp. No generic is present. **Use Objects, not primitives!** lastEl(1,2,3)

pub <T> T lastEl(List<T> ..) OK

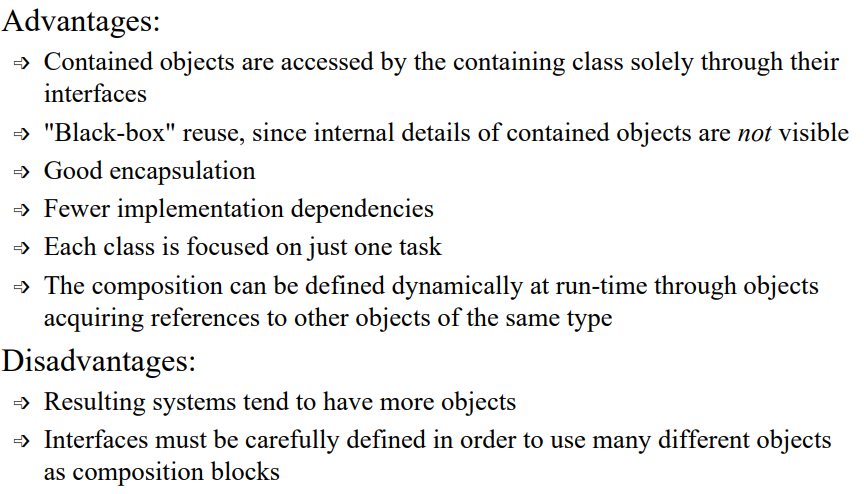
Integer[] nu={1,2,3}, lastEl(nu) OK

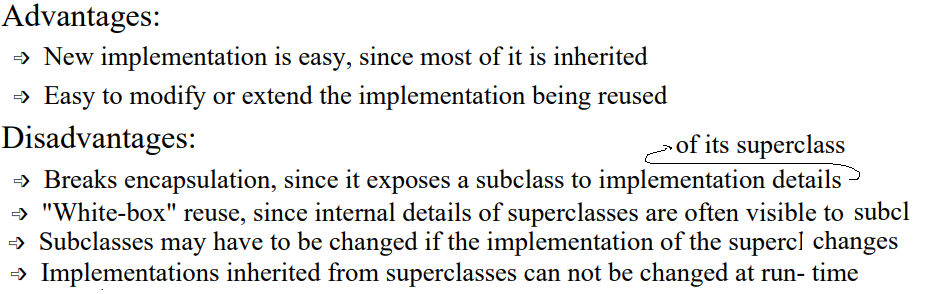
int[]num={1,2,3} NO. pub <T extends Number> T lastEl(List<T> ..), T is a Number or subcl. Void f (SomeClass<?> sc){}, f() is not generic. It is accepting SomeCl. But Somecl is generic! Can limit by SomeCl<? Extends Number>. Pub cl Arrls<T extends Number>{

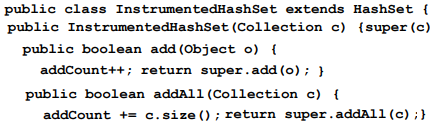
Pub T get(int ind){}}, in the Class, we define T ext Numb. We cant say T ext. Num. in get function. **Varargs** type... vars Any # of args will be accepted and put in vars[]. Use Objects, not primitive. **OO PRINCIPLES**

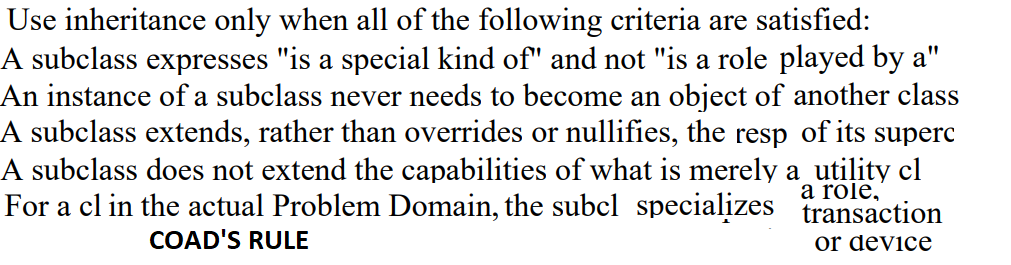
**1)Min. Acces. Of Class & members**

2)Favor Comp. Over inherit

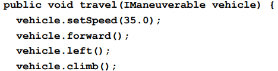






****

**3)Program To an Int. Not and Imple.**



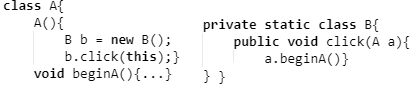
**4) OCP,Open for ext. Closed for Mod.**

**5)Single Choice Principle:**

**6)Liskov Substitution Principle:**

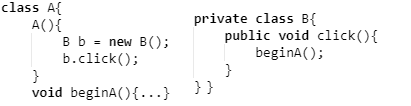
Functions that use base class must use subclasses without knowing it.

**INNER CLASSES Nested Class**

****

Normalde stat. Cl. init edilmez. Ama B burada A’yı modify edebilir. İnit şart. It will execute A through ref.(this). Priv. Olduğu için dışarıdan erişilmez. Pub olsa *A.B b = new A.B();* b.click(a)in içine A ref. göndermek lazım

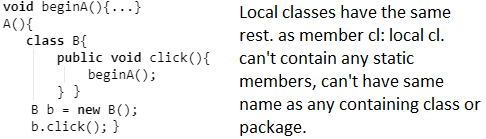
**Member Class**

****

This olmadan beginA’ya access yapabilir. Each instance B will belong to outer class. Outer class can access member clas’ members/instances. Reverse is also true. Member classes can't contain any static members, can't have the same name as any containing class or package. If the member class B public:

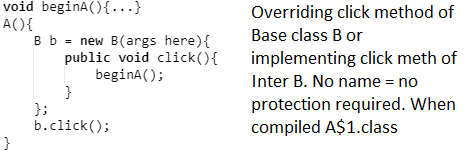
A a = new A(); A.B b = a.new B();

b. diyip A’nın metoduna erişemezsin. b.beginA() diyemezsin, b’de o yok. Ama b.click() diyebilirsin, A’ya erişebilirsin. **Local Class**

****

Sadece bu scope’ta geçerli. Ex: constr. dışına çıkınca B yok artık. For all 3 class, if you compile the code, you’ll have A$B.class. B inside A.

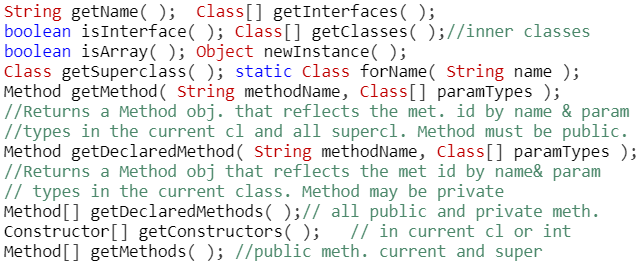
**Anonymous Class**

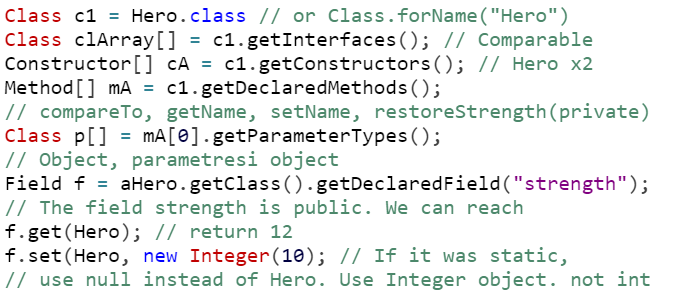


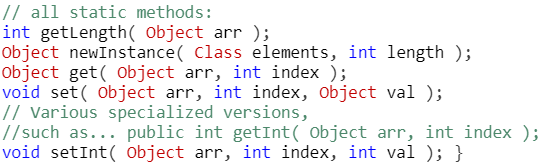
**Reflection CLASS**, obtaining a class



Methods in Class:

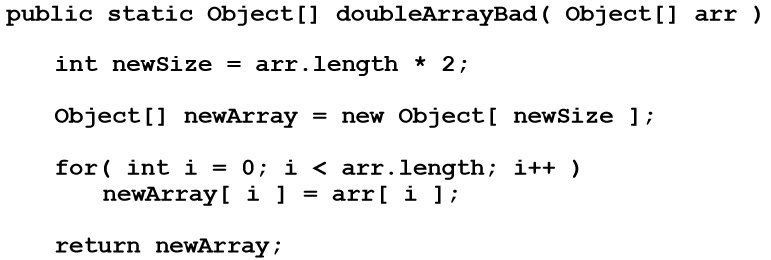


**ARRAY** All methods are static.

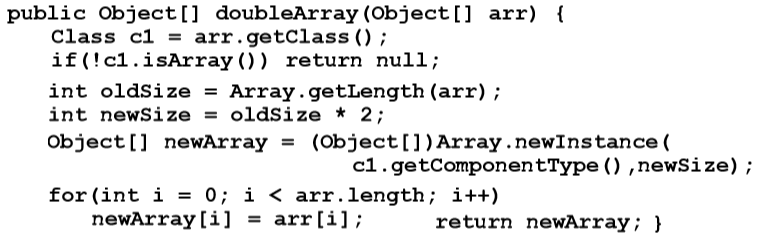


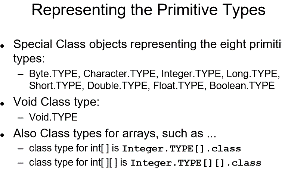
Creating Arr:

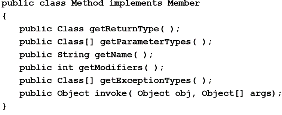
Ex: Double Arr.



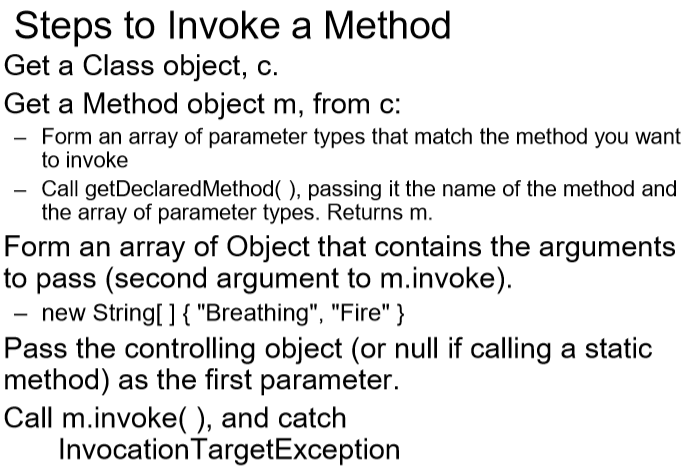
Won’t work. Because it accepts object array and copies objects to the new array. It doesn’t preserve types

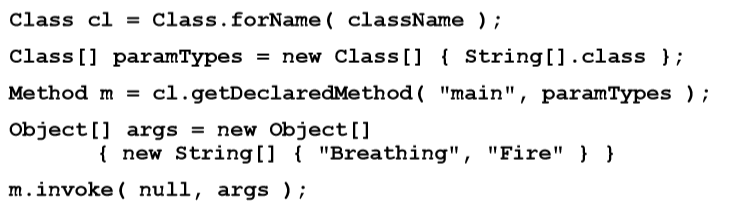
****

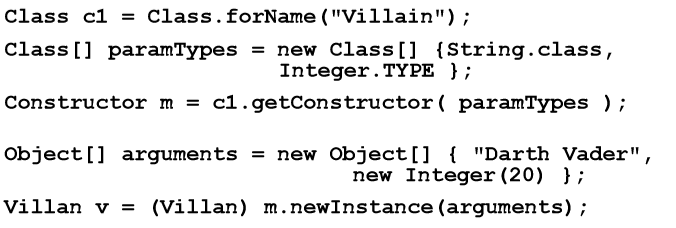


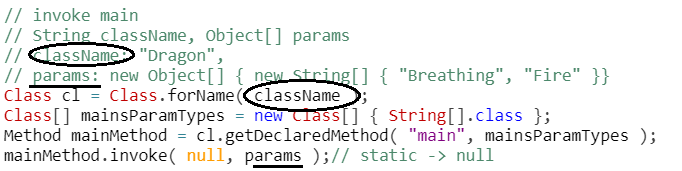
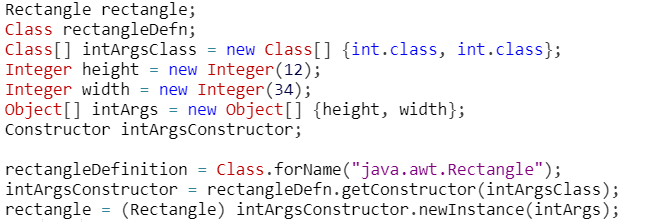


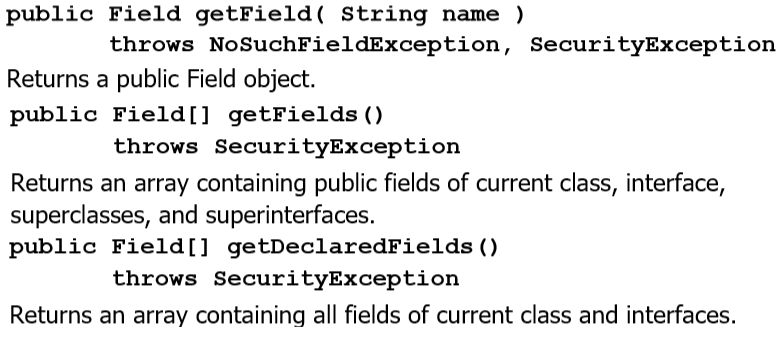
Obj[] args -> Integer.TYPE in an array

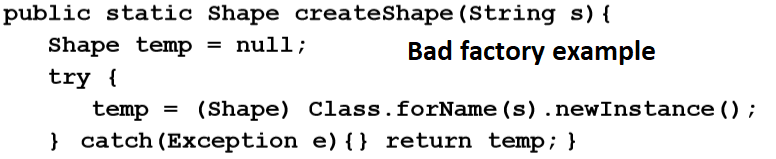
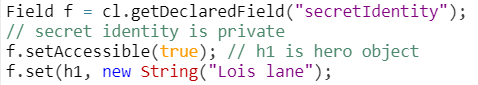
****

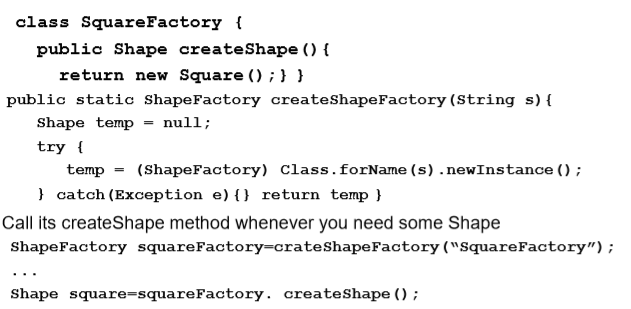
****

****



****

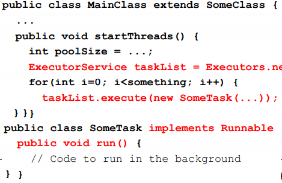




**MULTITHREAD PROGRAMMING**

****

To execute a class, it must implement Runnable. **Approach 1: Seperate Cl.**



taskList.execute(new Coutr(this, 6));

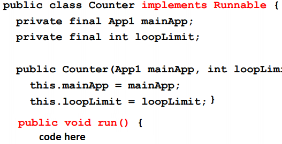
taskList.execute(new Coutr(this, 5));

Adv/Disadv:

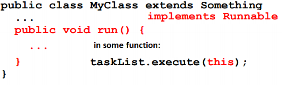
+ Can change independently

+ Can pass args to SomeTask.

- Race cond. – Hard do access main app(use this reference). Example:

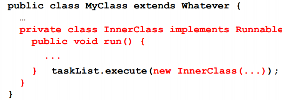


**Approach2:Main App Imp. Runnable**

****

Can run methods in this class, since it is in this class. Cant customize(no constructor, only this. Aynı tL.exe(this) yazarak çalıştırıyoruz. Run metod sadece bu classta, bu class’a özel. Race conditions. App3. both

**Approach3: Inner Cl. Imp. Runnable**

****

****

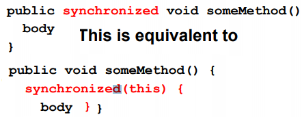
Can pass args. Easy to acces main app. Methods in inner class can acces any public/private methods in outerclass. –Race cond. –Run method tied to this app. **Approach4: Lambda**

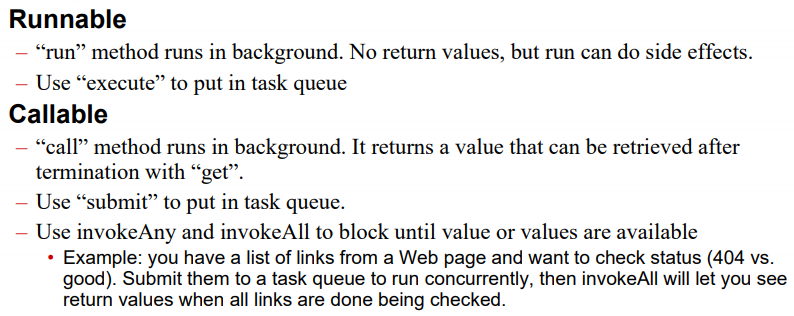
taskList.execute(()-> doSomeTask(..));

**Race Conditions**

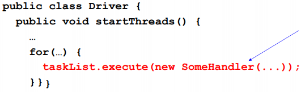
**synchronized(someObject) { code }**

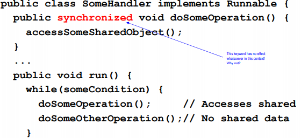
**Normal meaning**: Once a th. Enters that code, no other th. Can enter until 1st leaves. **Actual meaning:** Once a th. Enters that code, no other th. Can enter that code using the same someObj(lock object).



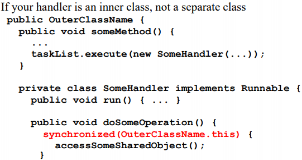


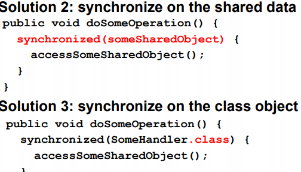
Use **volatile** keyword to make sure threads fetch that member from the memory. Not used for solving race conditions. **Sync. Problems:**



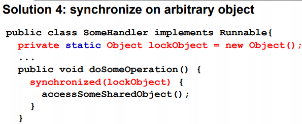


It does not work. Because in tasklist.exe, we are creating new inst. Every inst. has diff. lock obj. So, they wont wait for each other. **Solution1**:

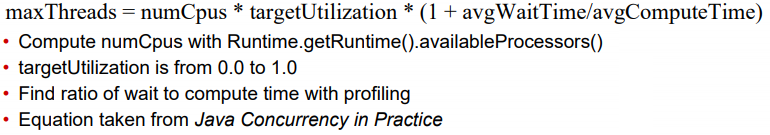


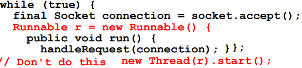


Note: If you use sync. For a static method, the lock corresponds to A.class object. Not this. So you will lock all of them.



**Determining Max. Thread Pool Size**

****

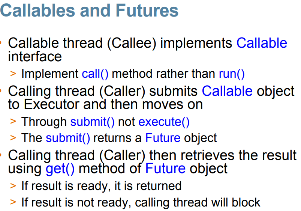


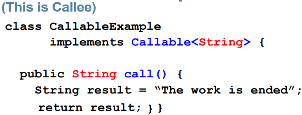
You have no control over the thread r.

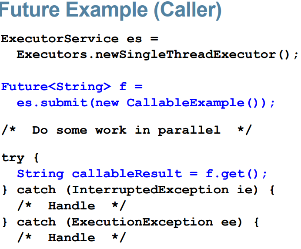
Instead, create pool, pool.execute(r). Now, you can use shutdown etc.

left fork, right compute, left join,

combine

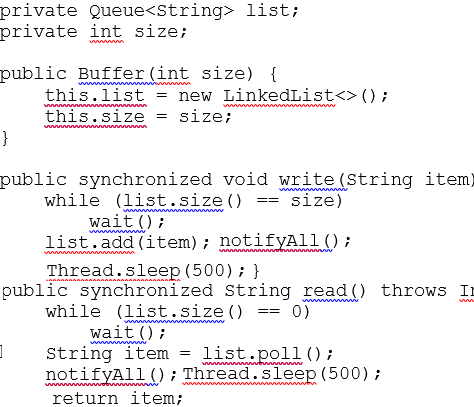




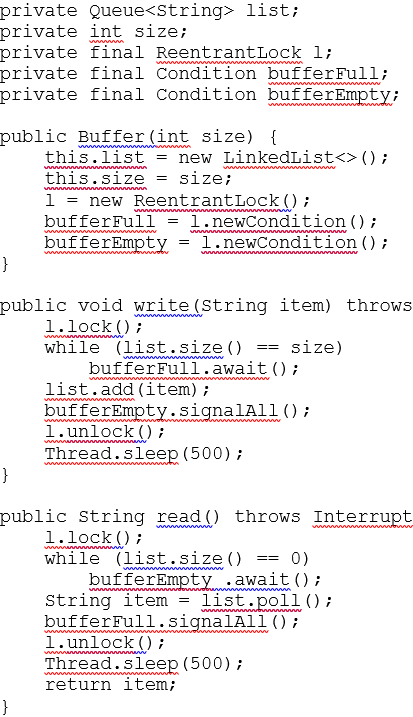


**Semaphores** Thread trying to access resource calls aquire() > Returns immediately if semaphore count > 0 > Blocks if count is zero until release() is called by different thread

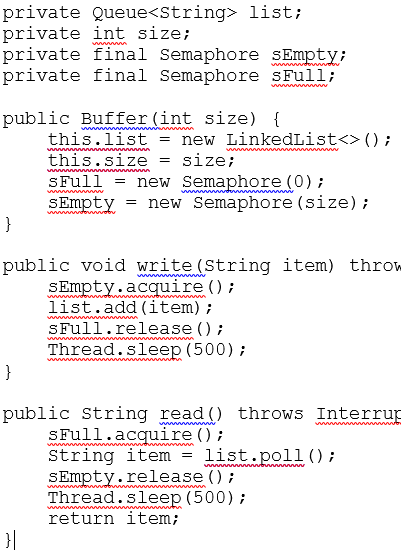
**Synchronized, wait&notify**



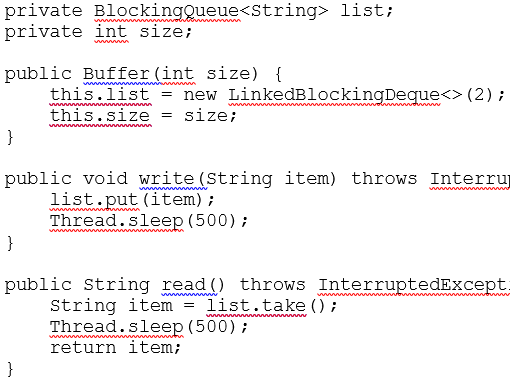
**Locks**



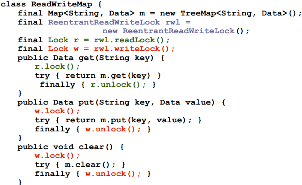
**Semaphore**

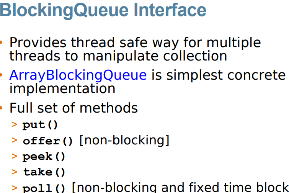


**Queue**

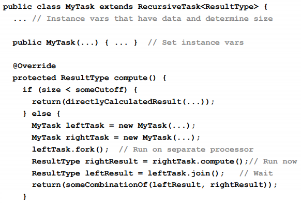


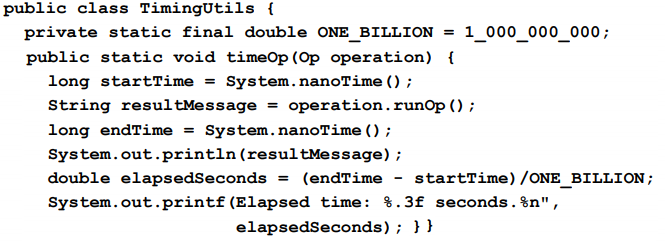
**ReadWrite lock**

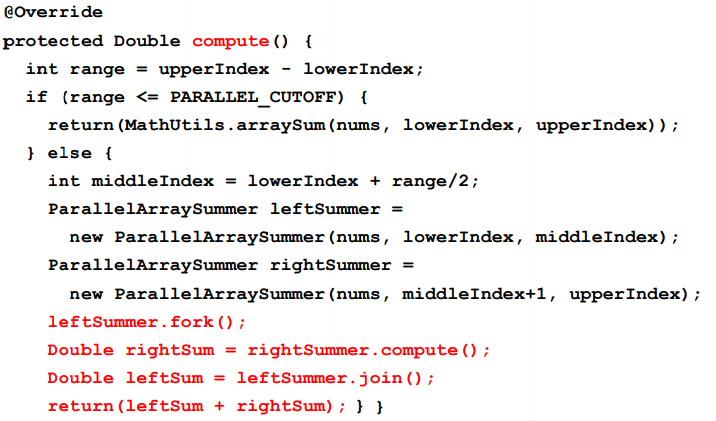




**Multithread**



****

****