Day1:

Loose Coupling

High Cohesion

Single responsibility principle

Open closed Principle

Liskov Substitution Principle

Dependency Inversion Principle

Interface segregation principle

Release reuse equivalence principle

Common closure principle

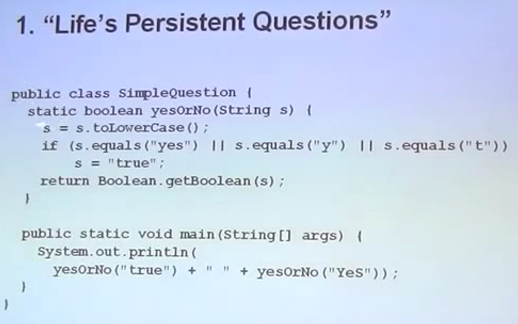
Common Reuse principle

Acyclic dependency principle

Stable dependency principle

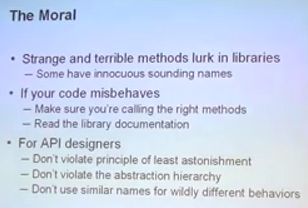
Stable abstraction principle

**Day 2**

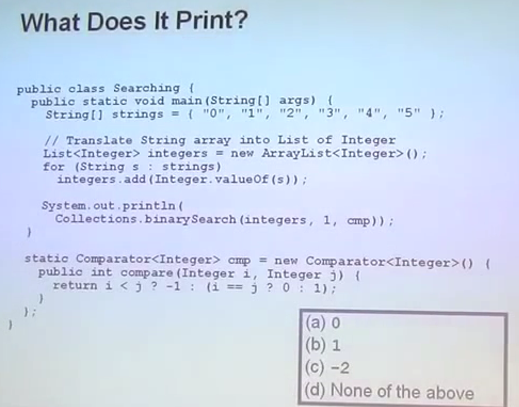
Q1) 

False false

Boolean.getBoolean returns true only if it can get the system property named by the argument exists and is equal to true. So Boolean.parseBoolean should be used instead.



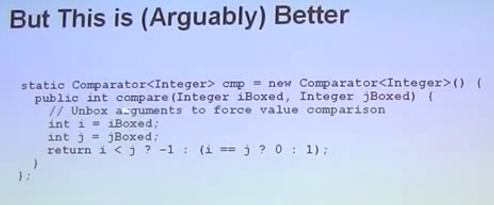
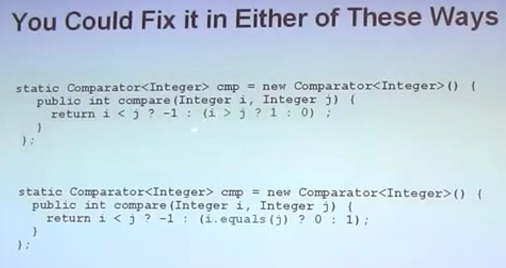
Methods should do what it’s name, arguments suggests. Should not astonish the users.

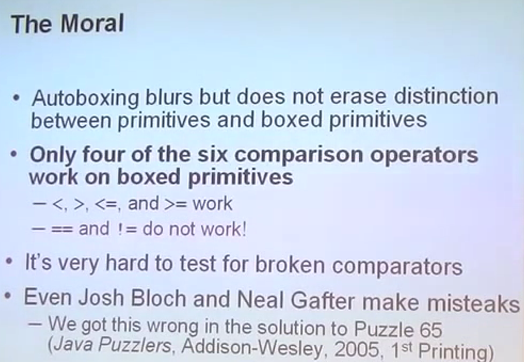


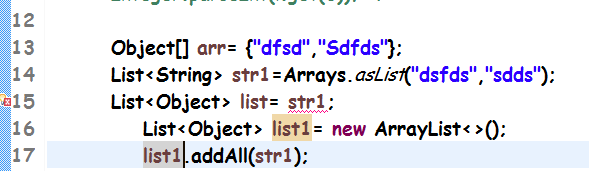
Comparator broken, autoboxing tricky

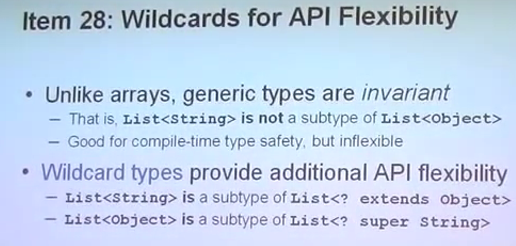
I==j: object references are compared.

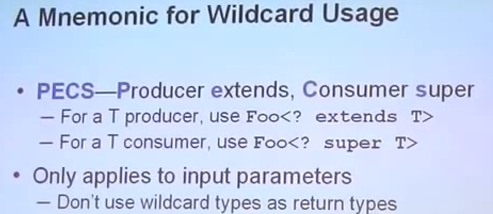
o/p: -2

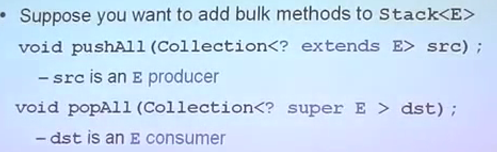


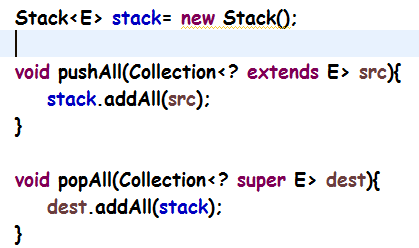




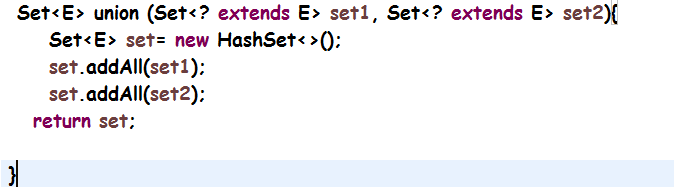


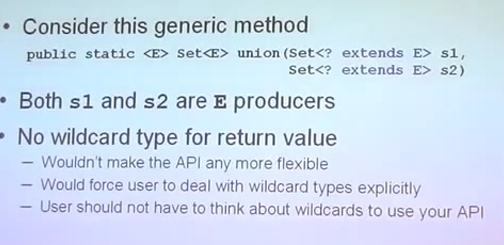


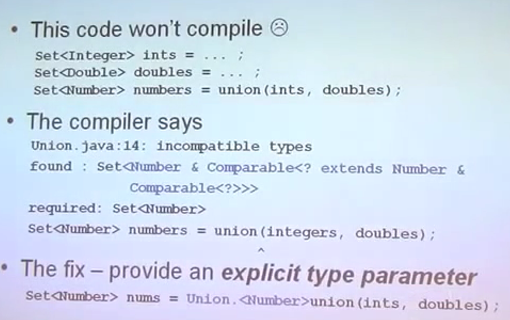


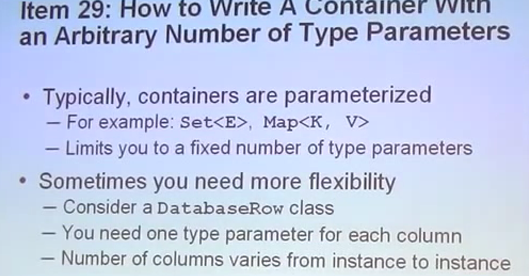


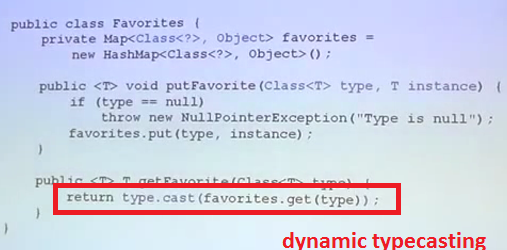
Don’t use mnemonics for the return type.

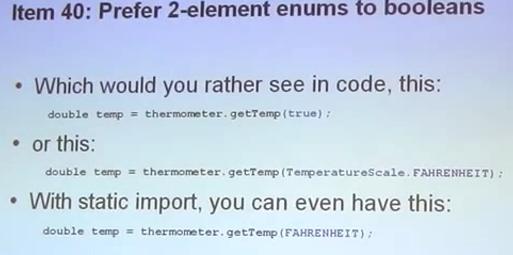


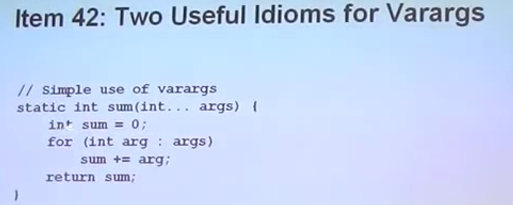


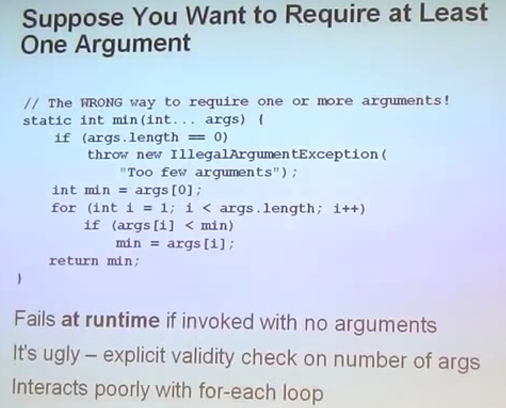


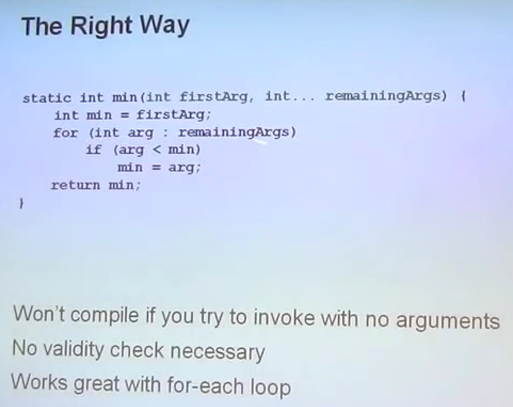


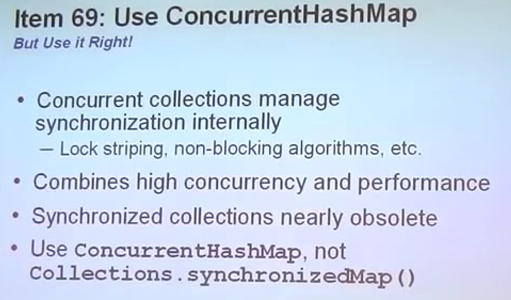


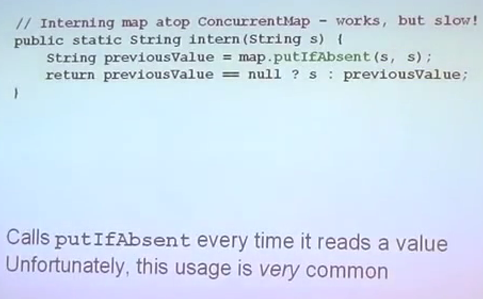


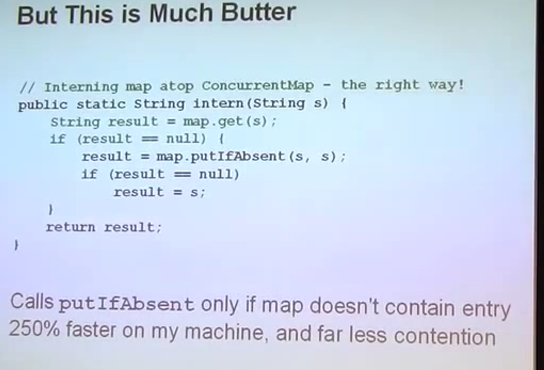


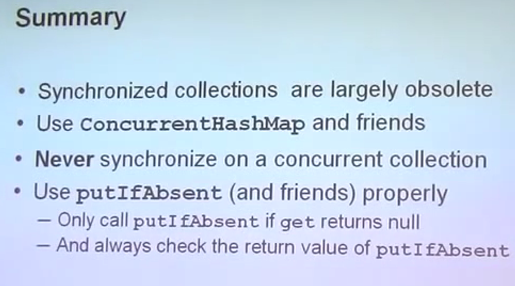


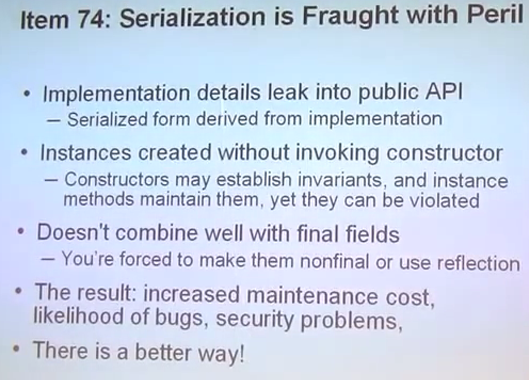


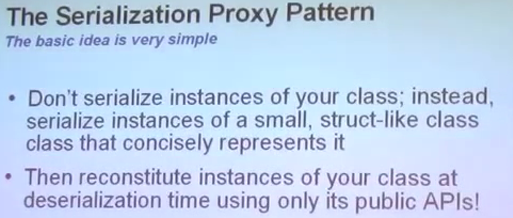


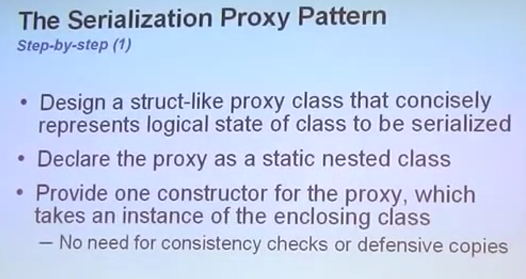


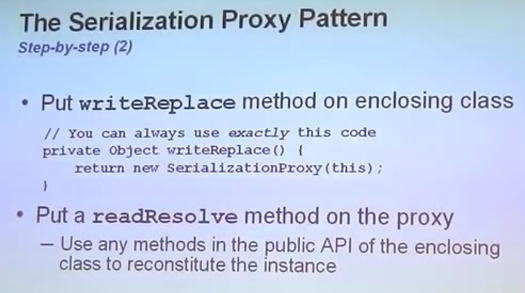


s

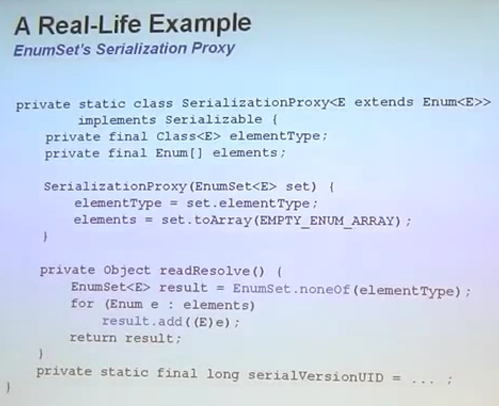








Instead of serializing object, serializes value returned by writeReplace()



JVM Architecture:

Virtual Machine: software simulation of a machine. Eg calculator software. JVM: Not physical machine.

1. Hardware based / System based VM: In a computer, several logical machines for different users. Eg: cloud computing, xen, vmware. Share hardware resources
2. Application based/ Process based VM: Has runtime engines to run a particular programming language applications.

(jvm) to run java based applications

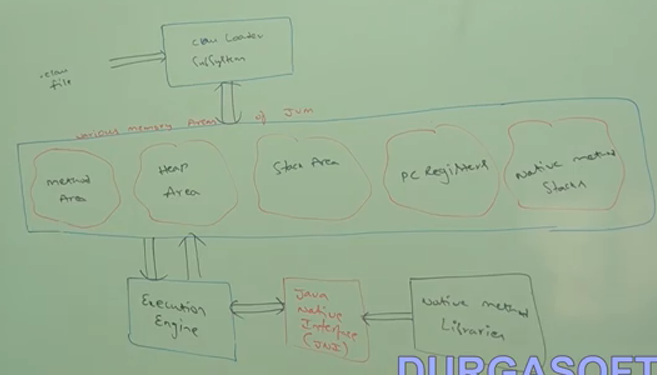
PVM: perl based vm

CLR: common language runtime: to run .net based applications

**JVM:** To load .class file (Class loader subsystem) and run these files. Part of jre. Execution engine to run .class file.

**Memory areas:**

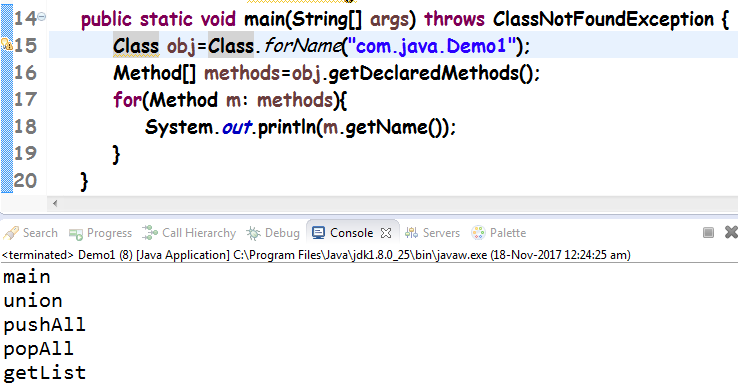
1. Method area
2. Heap
3. Stack
4. PC Register
5. Native method area
6. ClassLoader Subsystem: Loading (From hard-disk to jvm memory area (method area)), linking, Initialization
7. Memory area
8. Exection Engine
9. Java Native Interface



Virtual : Not original/ physical

In method area will have information about fully qualified name of the class, fully qualified name of the parent, methods information, variables information, constructors, modifiers information and constant pool information inside Class class object.

Heap area: Class class object created and stored here which has the class information.



If we have an object of a class, we can get the Class class object by calling .getClass() method.



**Linking**

1. Verify: Bytecode verifier. Process of ensuring that binary representation of a class is structurally correct or not i.e .class file is generated by valid compiler or not i.e it is properly formatted or not. Else java.lang.VerifyError if not correct.
2. Prepare: Static variables initialized to default values. In initialization phase, they would be set to original values. Static blocks executed.
3. Resolve