Object Model <java> Insight

By gbb21
MSTC.YQ

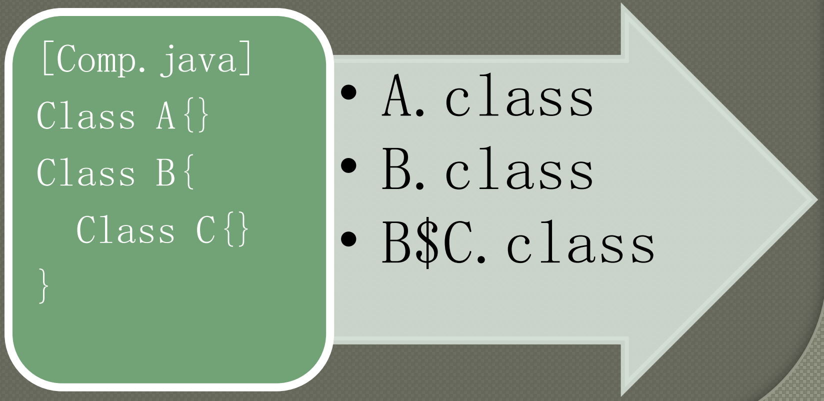
Classes - overall

- Life-time object oriented
- Not every thing is object
- Strong type and type erasure

Class file - Compile

```
ClassFile {
    u4 magic;
    u2 minor_version;
    u2 major_version;
    u2 constant_pool_count;
    cp_info
    constant_pool[constant_pool_count-1];
    u2 access_flags;
    u2 this_Class;
    u2 super_Class;
    u2 interfaces_count;
    u2 interfaces[interfaces_count];
    u2 fields_count;
    field_info fields[fields_count];
    u2 methods_count;
    method_info methods[methods_count];
    u2 attributes_count;
    attribute_info
    attributes[attributes_count];
}
```

- u1,u2,u4 are inner types of JVM,big-endian
- Keep all the meta data
- A class would be compiled into a class file, class structure could be persistently kept

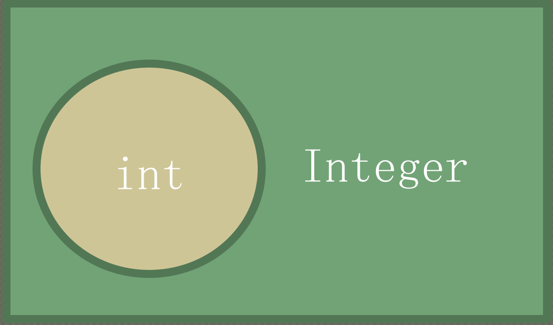


```
[Comp.java]
Class A{}
Class B{
  Class C{}
}
```

- A.class
- B.class
- B$C.class

Classes - primitive & Wrap

- primitive types
  - Allocated on stack
  - Fast speed
- wrapped types
  - Allocated on heap
  - Coherence
  - OOP traits



- size is fixed
- numeric type is signed
- Boxing and unboxing
- Example

```
int Val = 3;
ArrayList<Integer>
  List = new
  ArrayList<Integer>();
List.add(Val);
Val = List.get(0);
```

Classes - traits & difference

```
class T {
    Final int i = 0; // instance variable
      initialization
    static int version = 3; // class variable
      init
    String id = "x432"; // initialize object
      variable
    static int[] a = new int[100];
    Static { // Static initialization
      For(int i = 0; i < 100; i++)
        a[i] = i;
    }
    Public T(int Arg) {}
    Public T(){
      this(3);  // Call other constructor
    }
}
```

- Everything is in the class
- All the member variable could be initialized to zero
- Static block could run a piece of code to initialize static variable before construction
- Could call overloading constructor in constructor

Access rights (Java VS C#)

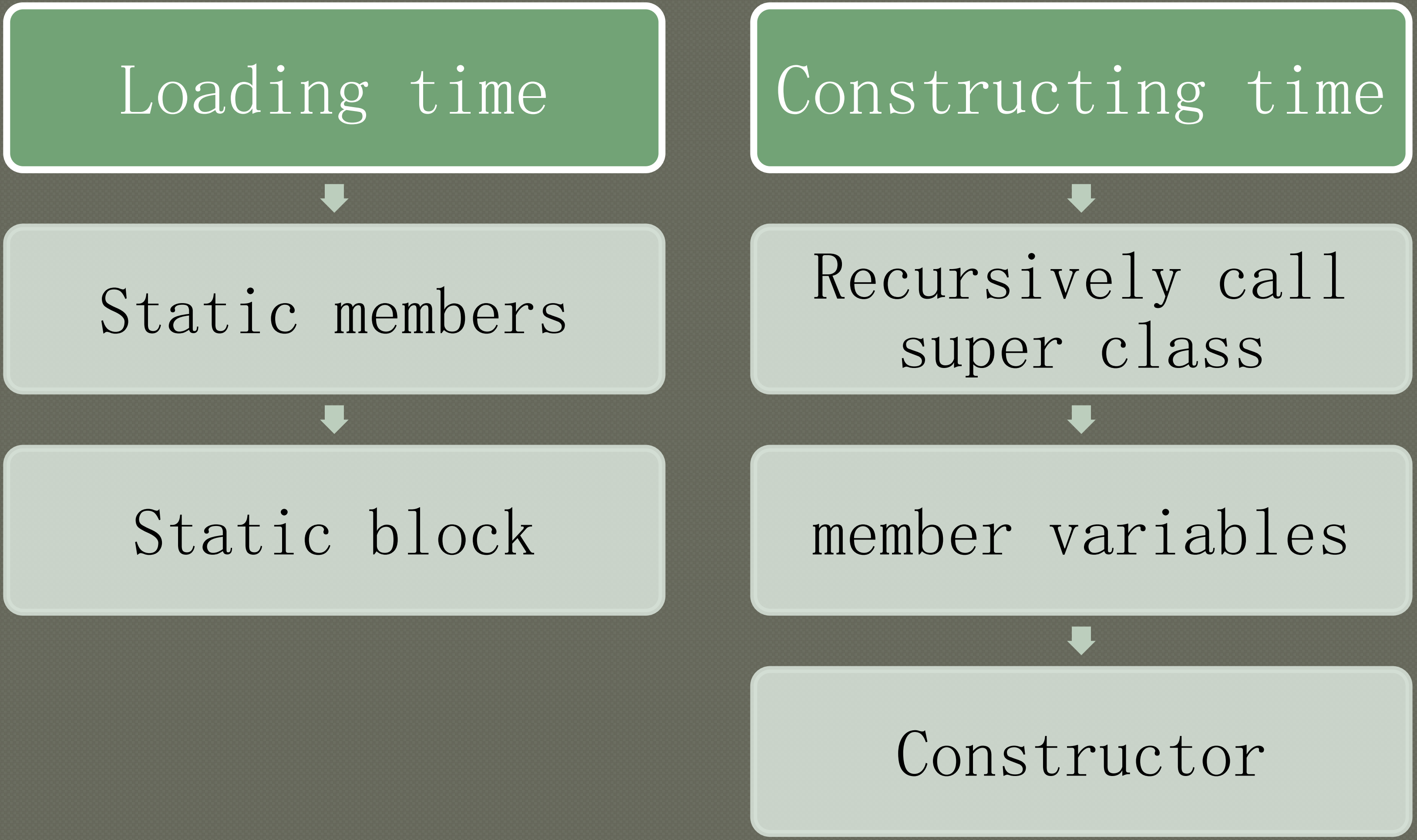
Class\member	Private	(non-modified)	Protected	public
(non-modified)	Only in the class itself	In the package	In the package	In the package
public	Only in the class itself	In the package	In the package; Derived classes out of package	Anywhere

Class\member	Private	Protected	Internal	protectedinternal	public
internal	Only In the class itself	The derived classes in assembly	In the assembly	In the assembly	In the package
public	Only In the class itself	The derived classes	In the assembly	In assembly; derived classes out of assembly	Anywhere

Classes - Inheritance

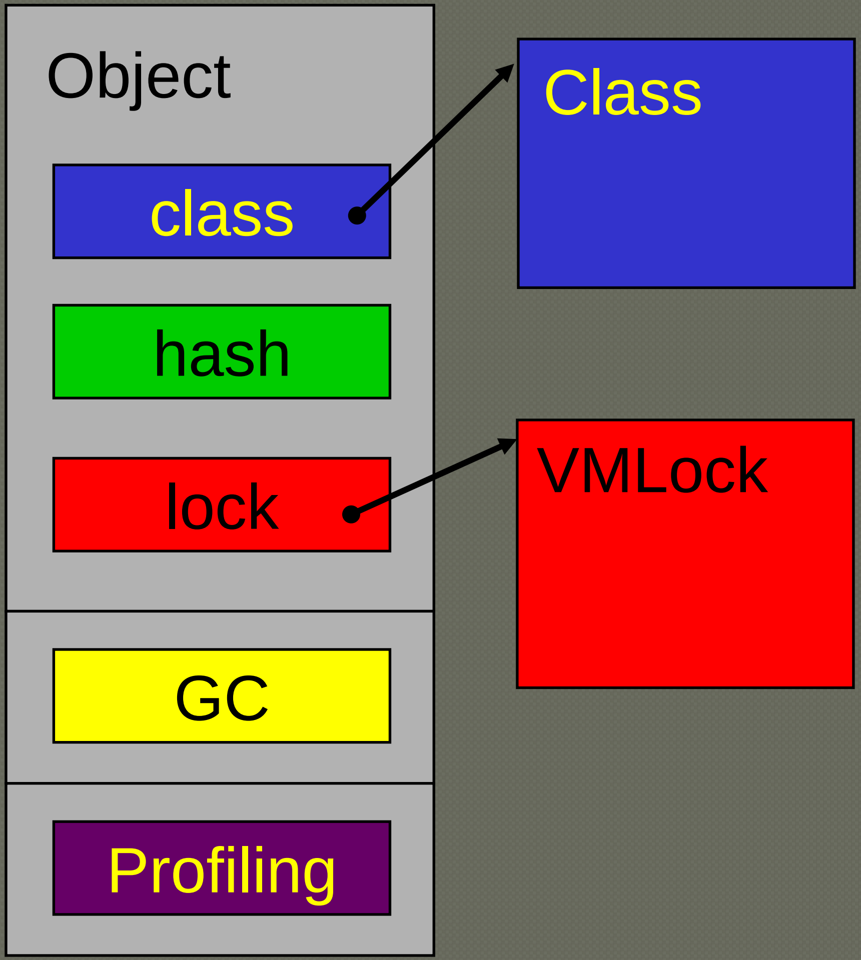
- Single inheritance model [FFC#]
- All classes are derived from “Object” [FFC#]
- All functions in classes are “virtual”
- Inner classes must be instanced with the pointer to outer object

Initialization order

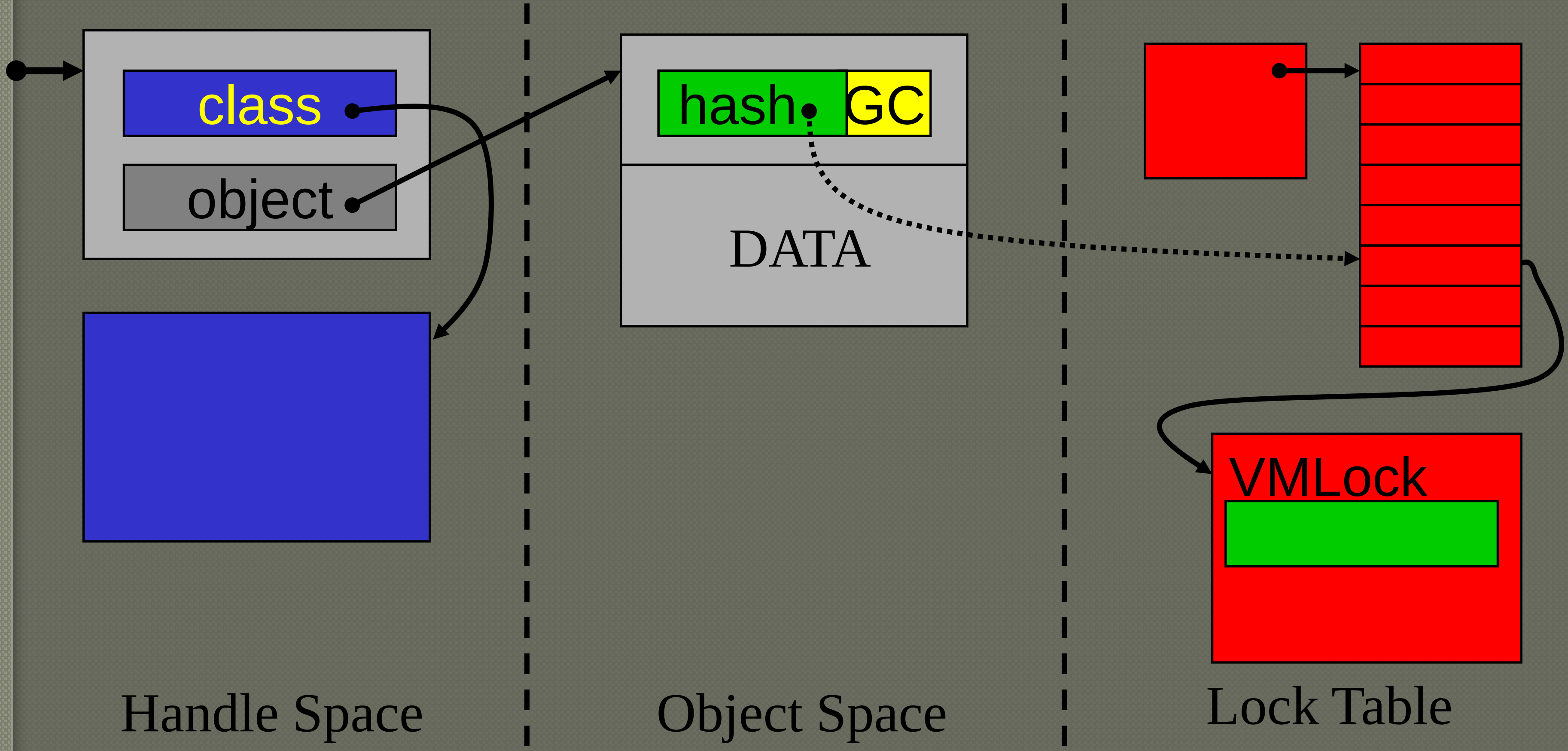


Object memory layout

```
class Object {
    Class getClass();
    int hashCode();
    void wait();
    void wait(long);
    void wait(long,int);
    void notify();
    void notifyAll();
    Object clone();
    boolean equals();
    void finalize();
}
```

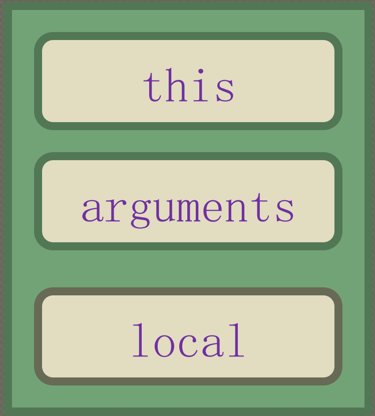


Original Sun Object Model



Java Stacks

- The JVM is very much stack-oriented.
- A stack frame is subdivided into two parts
  - a Local Variables section:
    - store all the local variables and arguments
  - an Operand Stack section:
    - method’ s instructions operate here.
    - Almost all JVM instructions are stack-based;
    - Example an “add” instruction pops the top two elements of the stack, adds them, and pushes the sum back onto the stack.



Heap and GC

GC OVERVIEW

- Incremental collection
- Trace all the available object from reference tree (Not reference counting)
- Avoid the circulate reference
- High cost of the collecting operation

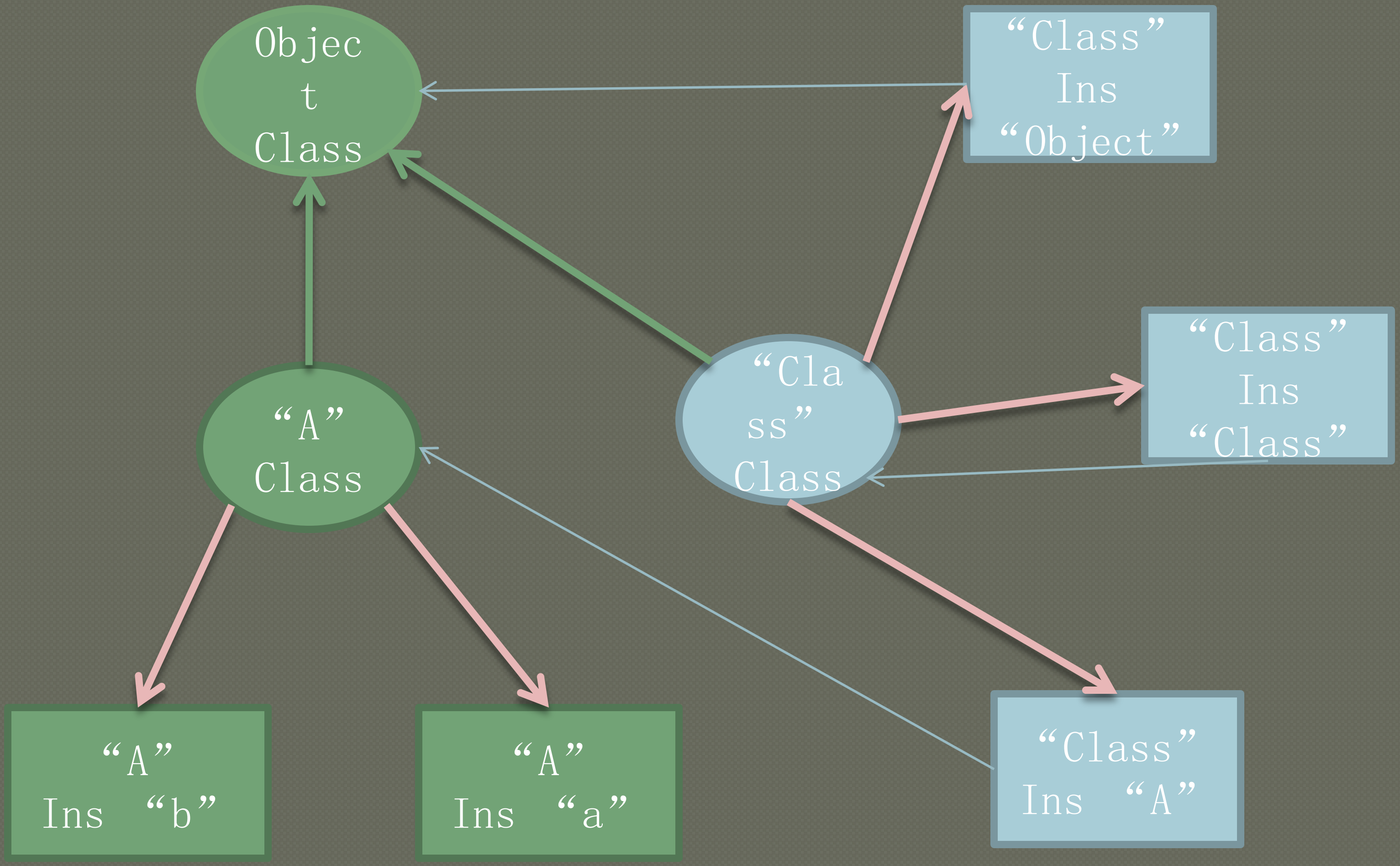
REFERENCE INTENSITY

Strong
- Unreachable object

Weak
- Soft reference
- Weak reference
- Phantom Reference

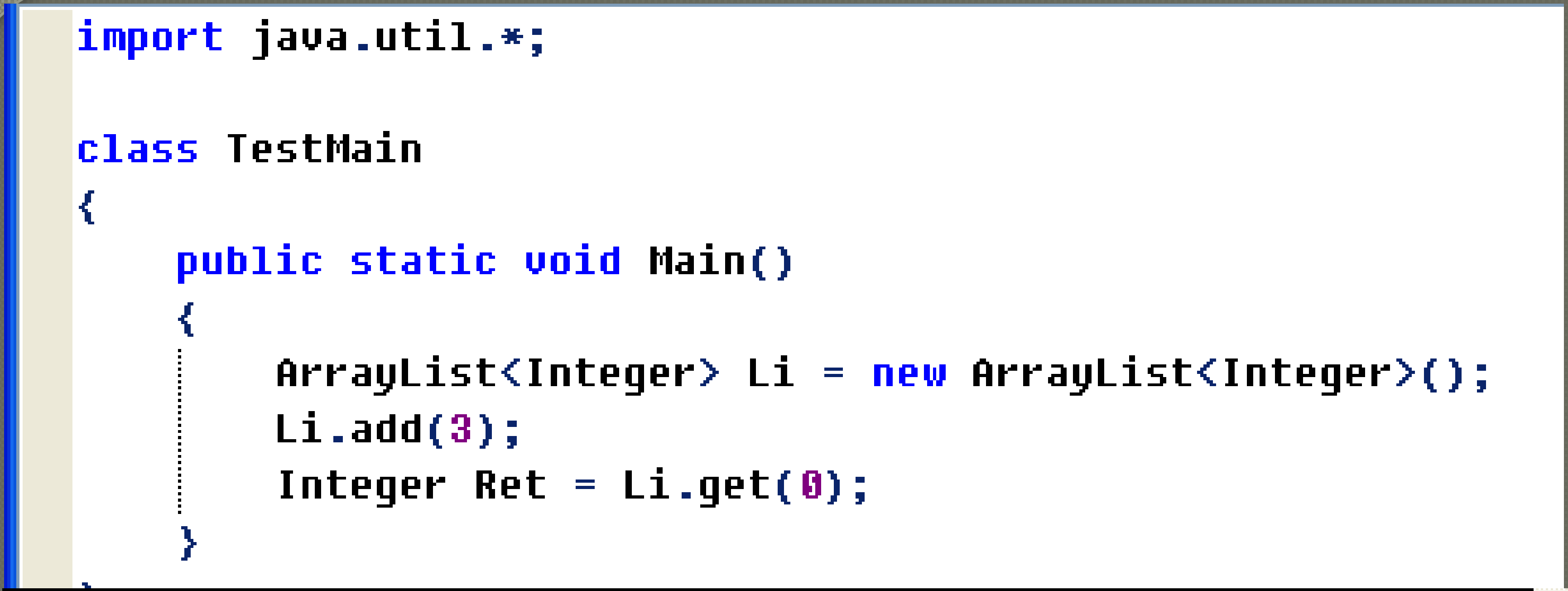
non
- Unreachable object

MetaClass(Class) model

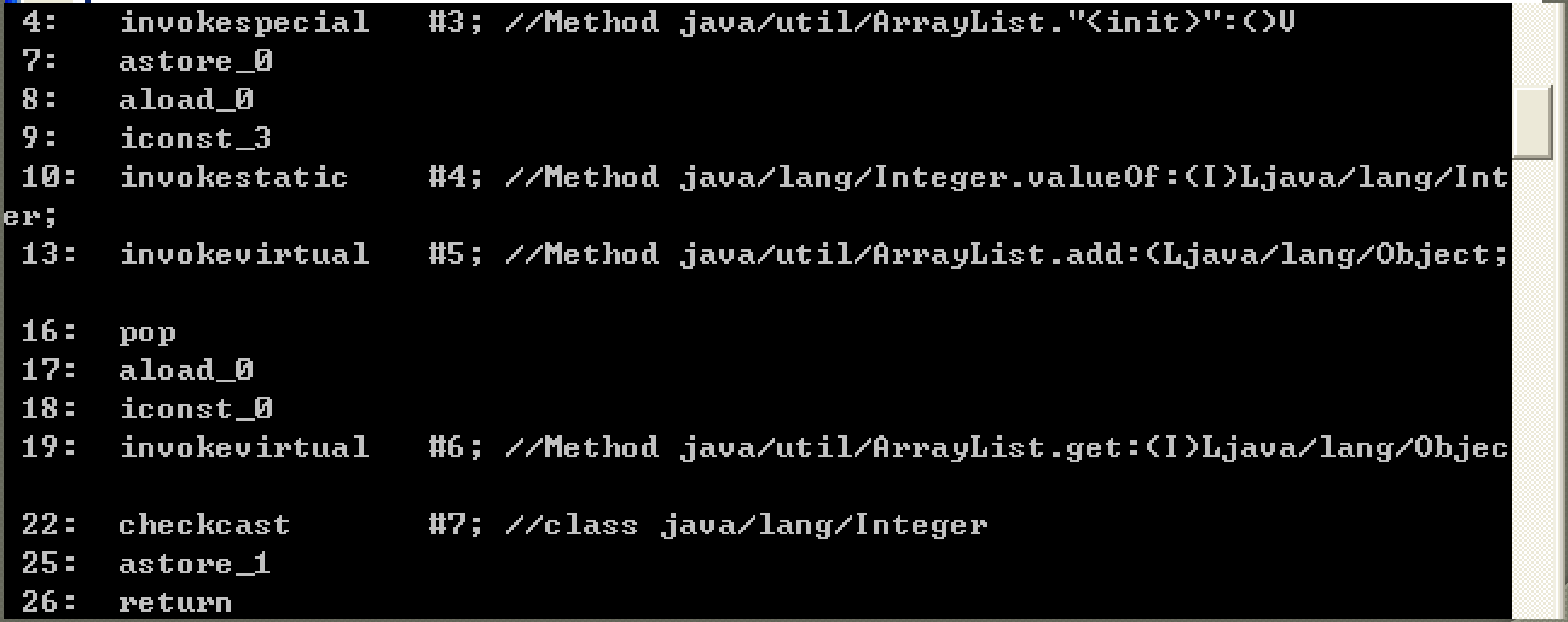


RTTI & GP

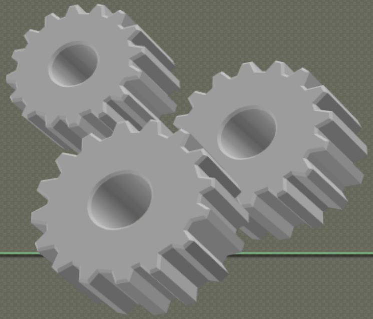
- "Class" object
  - "Object" class has a static pointer to a "Class object"
  - All of the type conversion will be checked by RTTI (throw exception)
  - The generic programming of java is implemented by RTTI
    - ArrayList<Integer> Arr = new ArrayList<Integer>();
    - Integer Var = Arr.get();
    - ArrayList Arr = new ArrayList();
    - Integer Var = (Integer) Arr.get();



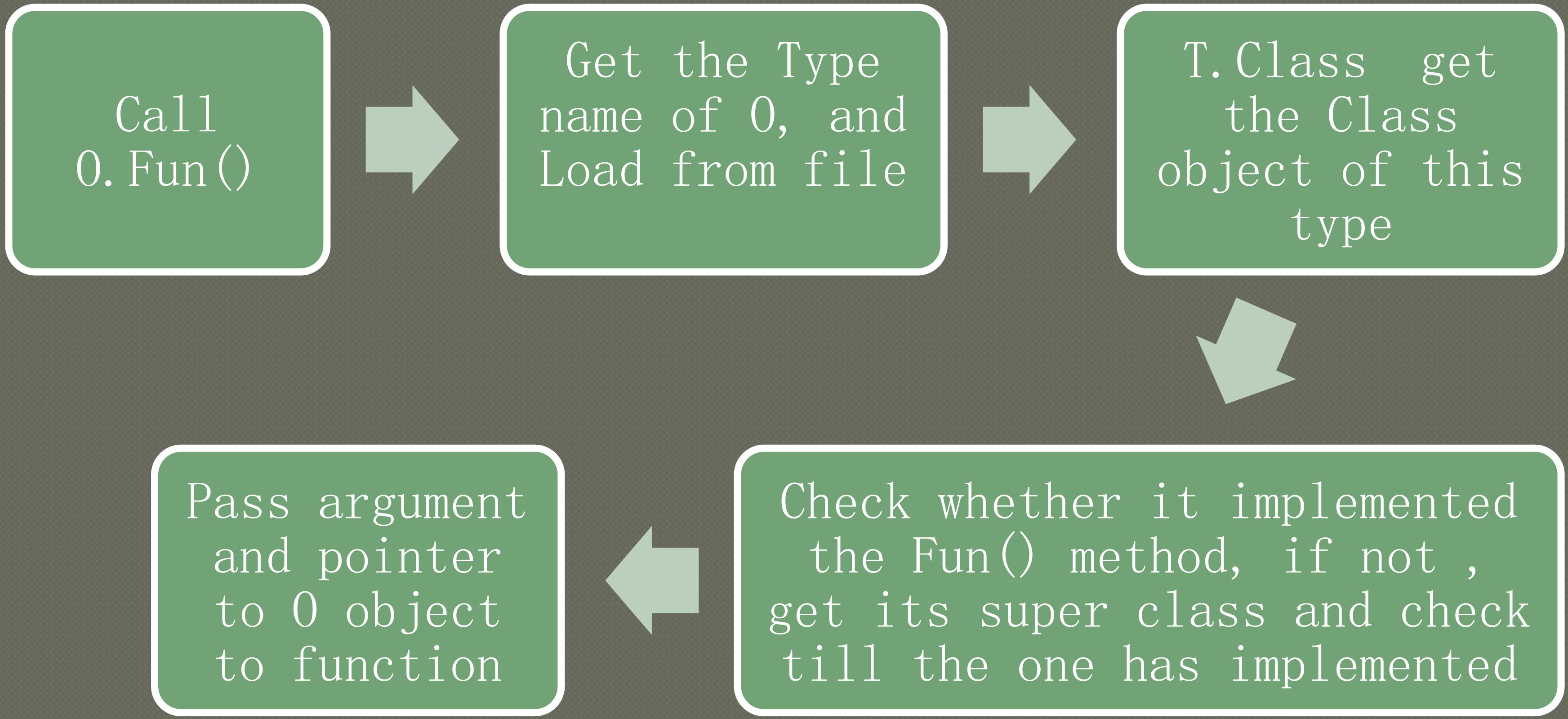
```java
import java.util.*;

class TestMain
{
    public static void Main()
    {
        ArrayList<Integer> Li = new ArrayList<Integer>();
        Li.add(3);
        Integer Ret = Li.get(0);
    }
```



```
 4:   invokespecial   #3; //Method java/util/ArrayList."<init>":()V
 7:   astore_0
 8:   aload_0
 9:   iconst_3
 10:  invokestatic    #4; //Method java/lang/Integer.valueOf:(I)Ljava/lang/Int
er;
 13:  invokevirtual   #5; //Method java/util/ArrayList.add:(Ljava/lang/Object;

 16:  pop
 17:  aload_0
 18:  iconst_0
 19:  invokevirtual   #6; //Method java/util/ArrayList.get:(I)Ljava/lang/Objec

 22:  checkcast       #7; //class java/lang/Integer
 25:  astore_1
 26:  return
```

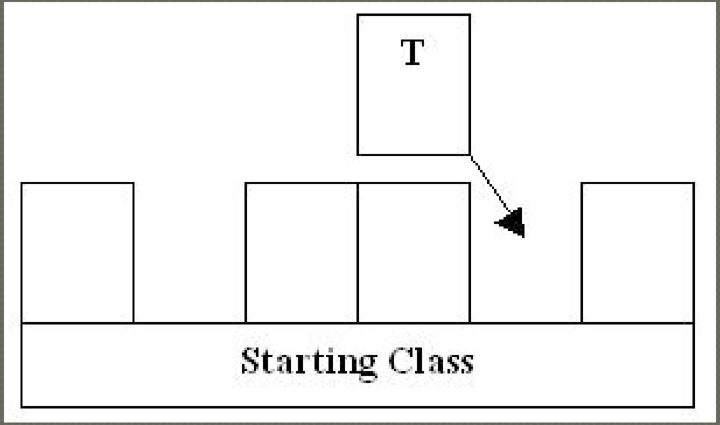


Polymorphism


Call O.Fun()
Get the Type name of O, and Load from file
T.Class get the Class object of this type
Check whether it implemented the Fun() method, if not , get its super class and check till the one has implemented
Pass argument and pointer to O object to function

Dynamic loading & JIT

- The first time use of some class, the class is loaded from .class file
- Load only what you need, save memory
- Runtime link, every class or component could be replaced easily
- Select piece of code and compile it for faster speed


T
Starting Class