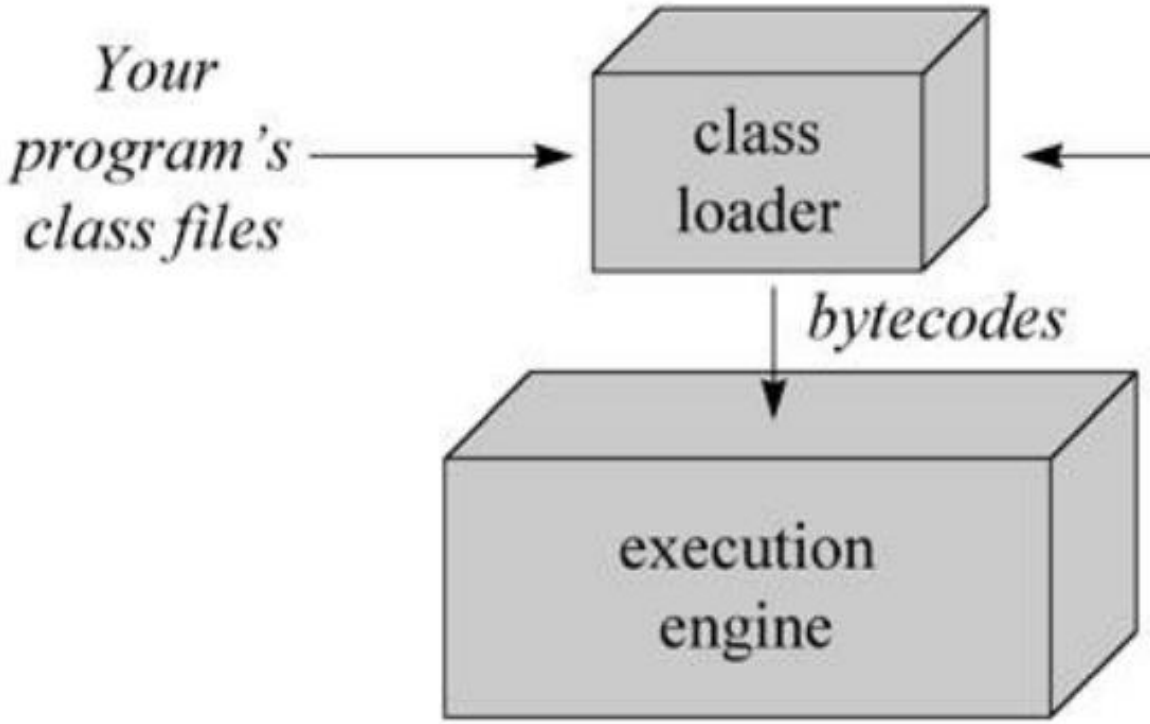
# JVM

A Java Virtual Machineís main job is to load class files and execute the bytecodes they contain.




a **class loader**,
which loads class files from both the program and the Java API

**The bytecodes are executed in an execution engine**,
which is one part of the virtual machine that can vary in different implementations

# The Class Loader Architecture

two types of class loaders

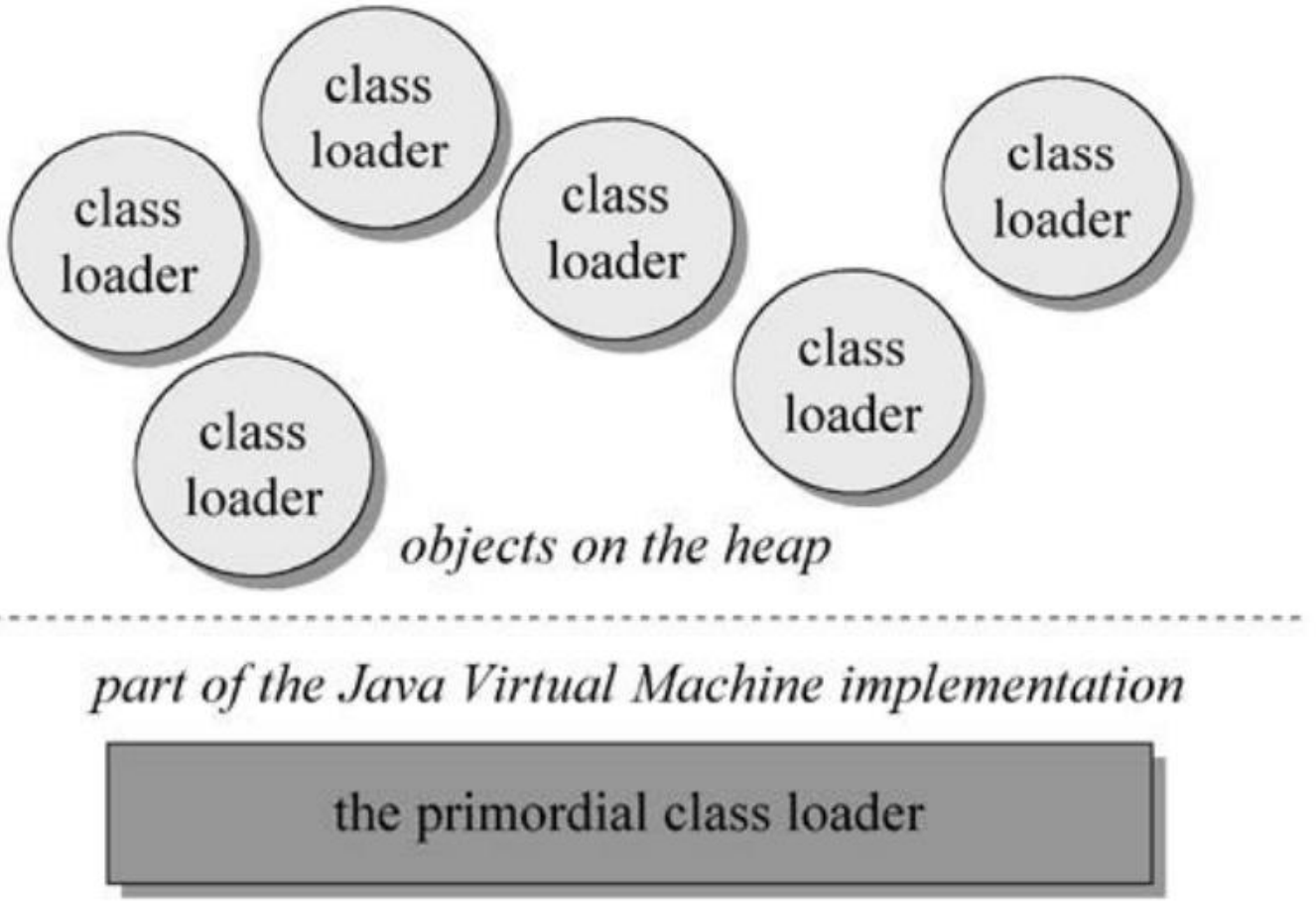
**1 class loader objects.**

- written in Java,
- compiled to class files,
- loaded into the virtual machine,
- instantiated just like any other object.

**2 a "primordial" class loader**

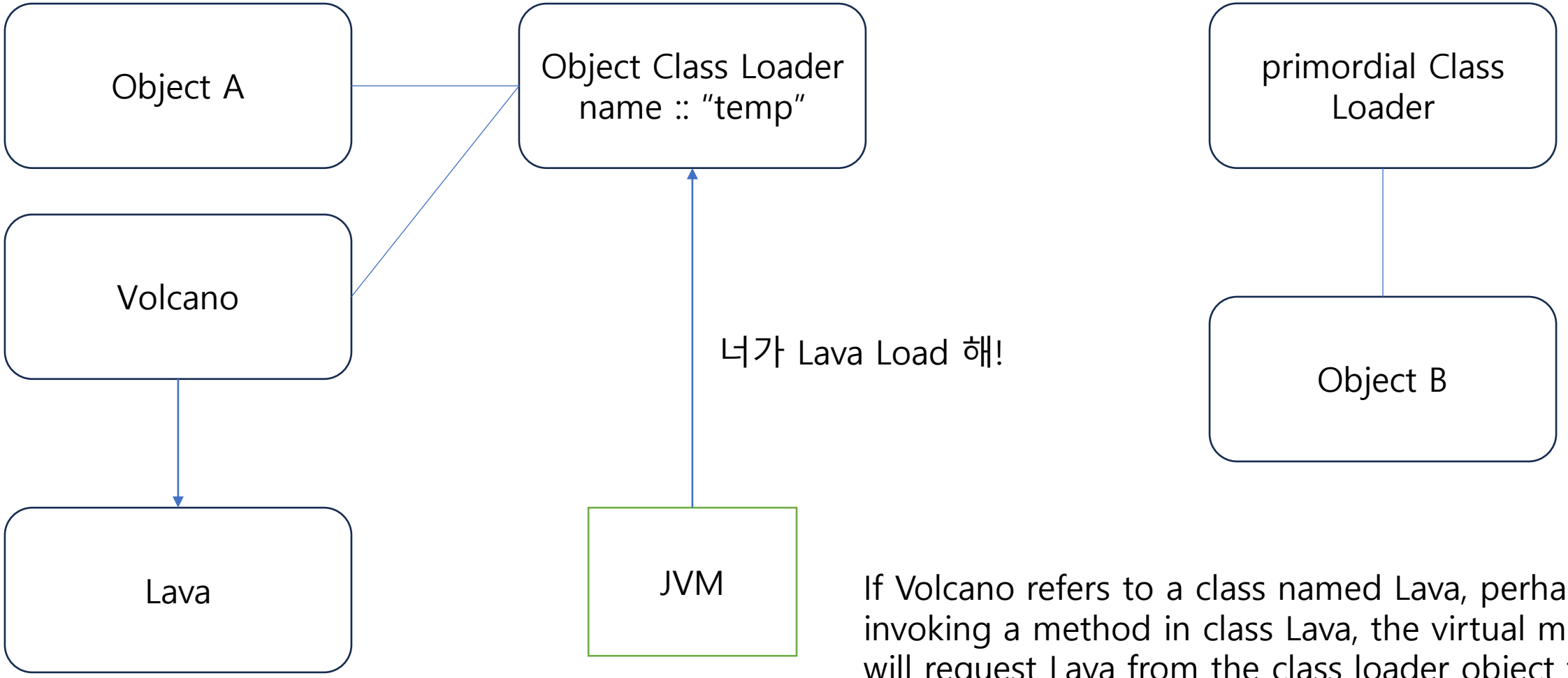
- primordial class loader loads trusted classes, including the classes of the Java API

They enable you to dynamically extend a Java application at run-time.



# The Class Loader Architecture

For each class it loads, the Java Virtual Machine keeps track of which class loader--whether primordial or object--loaded the class.

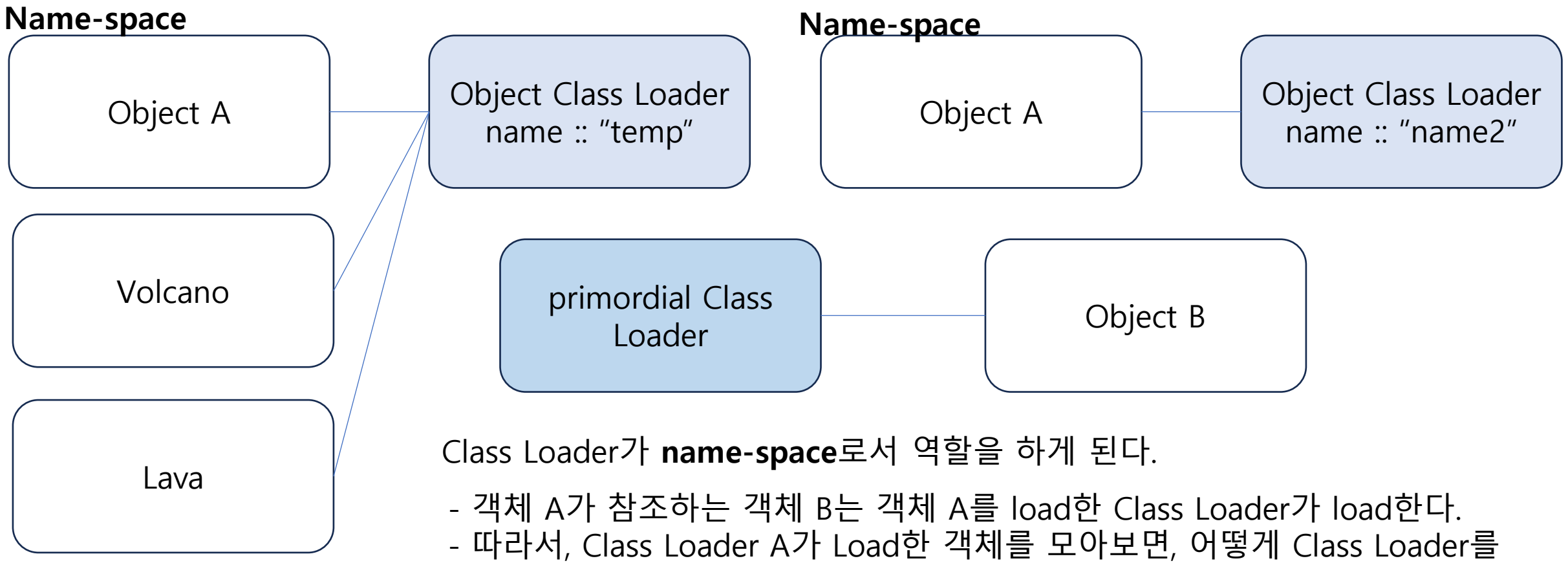



If Volcano refers to a class named Lava, perhaps by invoking a method in class Lava, the virtual machine will request Lava from the class loader object that loaded Volcano.

# The Class Loader Architecture

Because the Java Virtual Machine takes this approach to loading classes, classes can by default only see other classes that were loaded by the same class loader

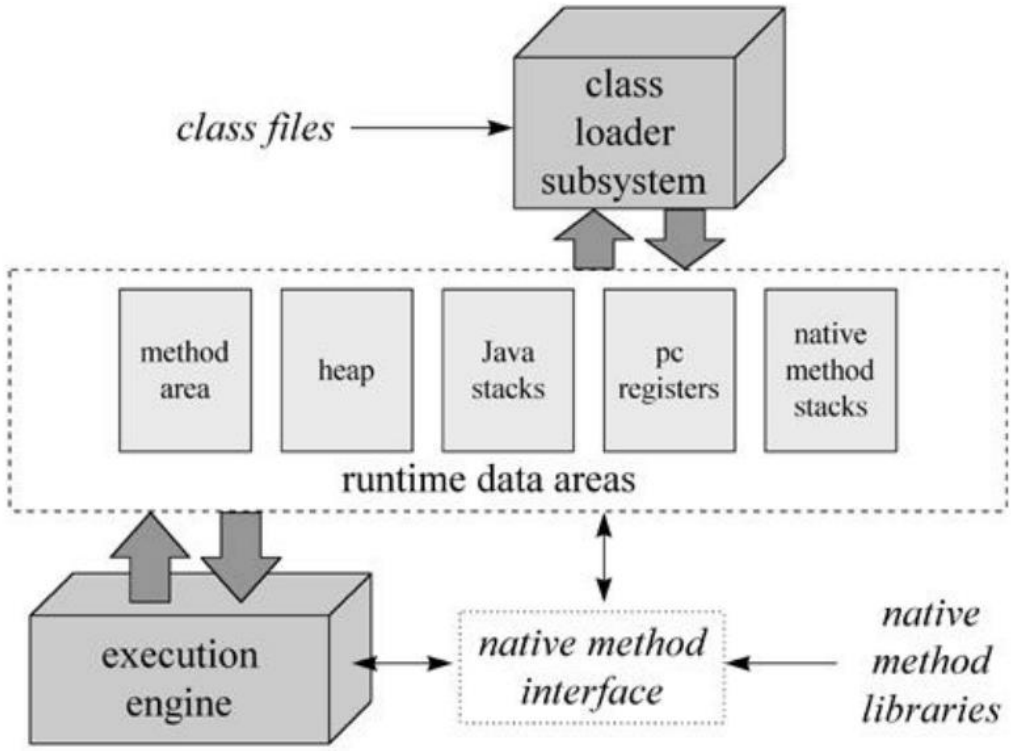
This is how Javaís architecture enables you to create **multiple name-spaces** inside a single Java application

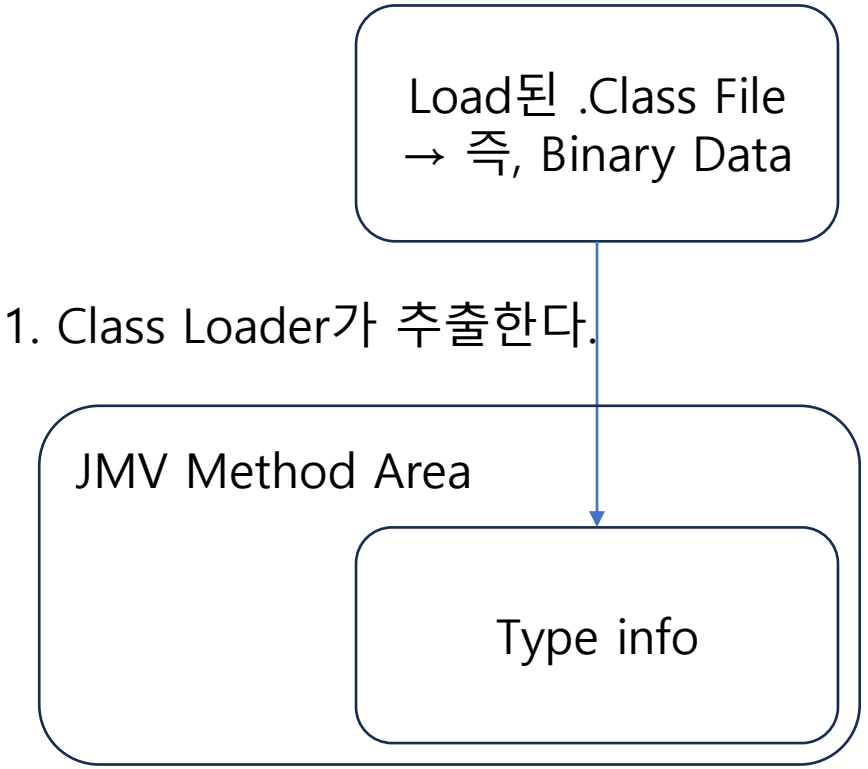



Class Loader가 **name-space**로서 역할을 하게 된다.

- 객체 A가 참조하는 객체 B는 객체 A를 load한 Class Loader가 load한다.
- 따라서, Class Loader A가 Load한 객체를 모아보면, 어떻게 Class Loader를 통해 name-space를 형성하는지 이해할 수 있다.

# The Architecture of the Java Virtual Machine

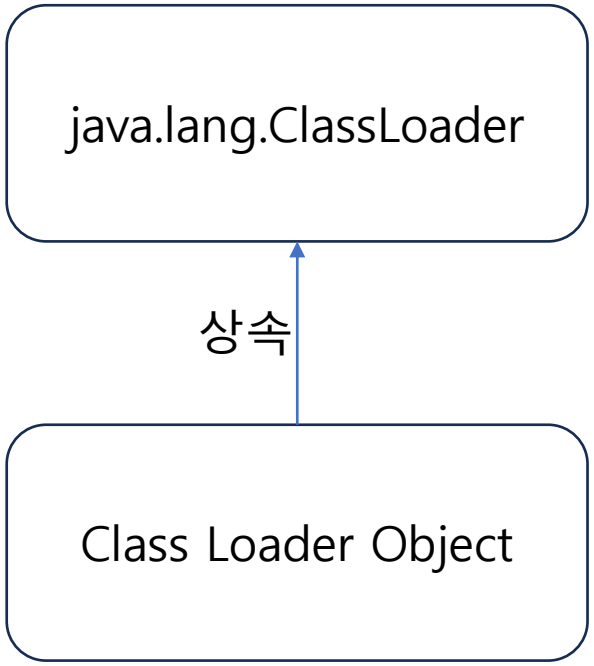



Step1.
When the virtual machine loads a class file, it parses information about a type from the binary data contained in the class file

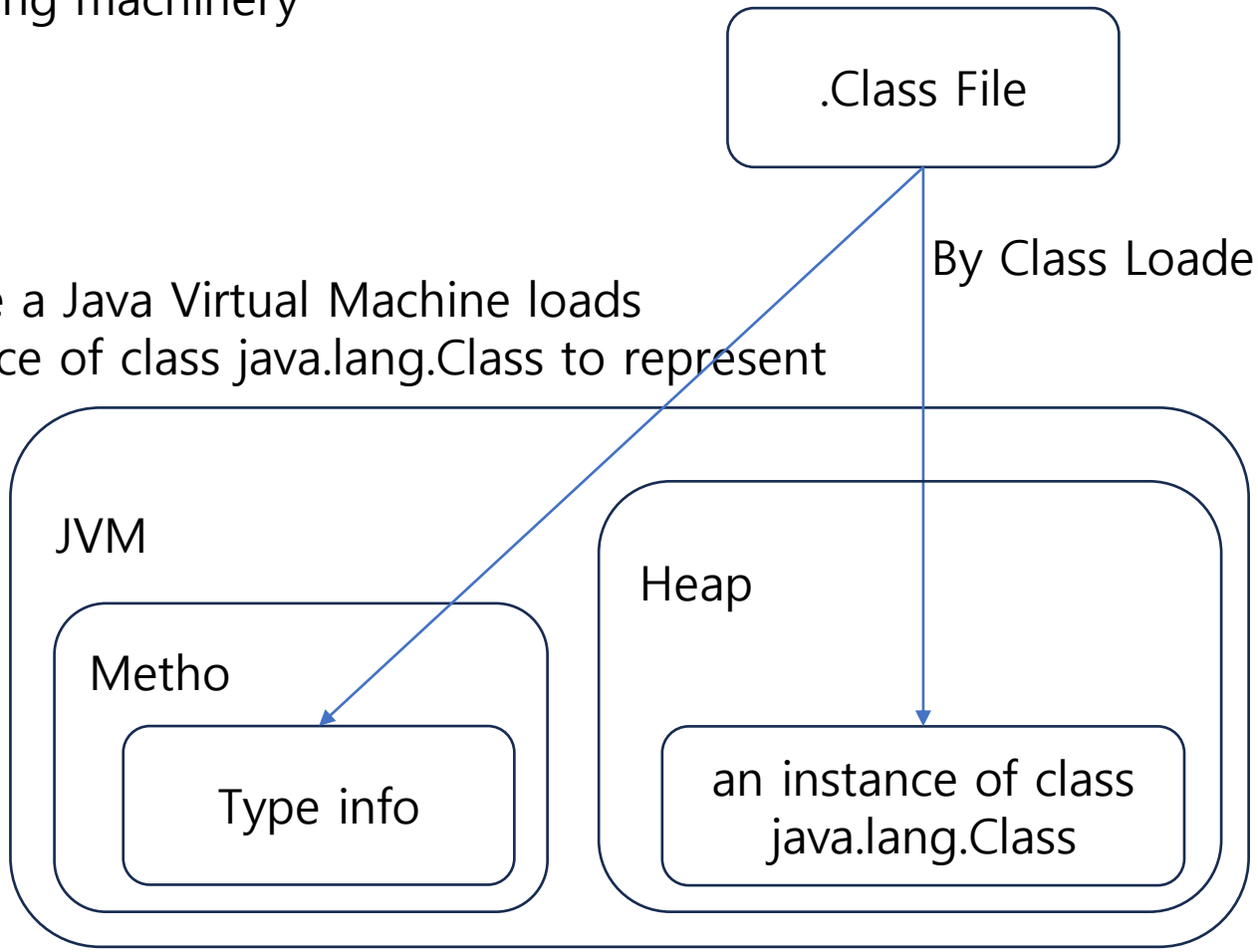
Step2.
It places this type information into the method area

# The Class Loader




The methods of class ClassLoader allow Java applications to access the virtual machineís class loading machinery

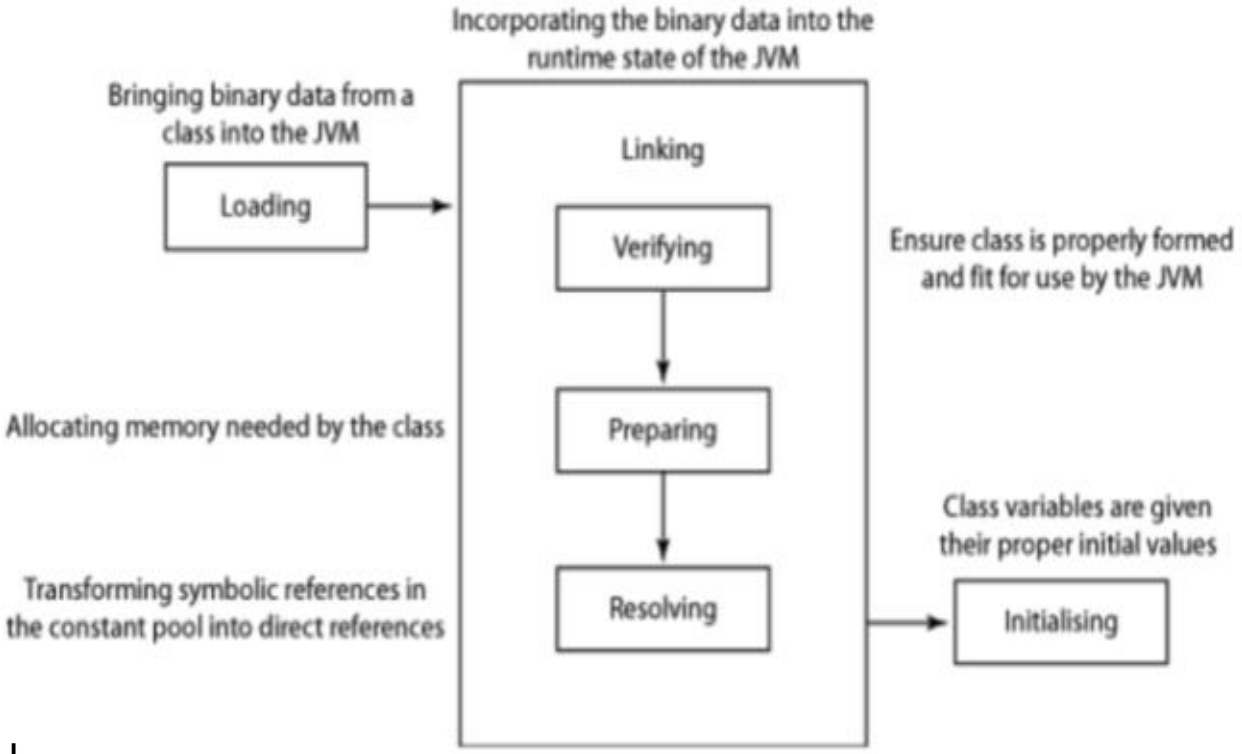
Also, for every type a Java Virtual Machine loads it creates an instance of class java.lang.Class to represent that type.




Like all objects, class loader objects and instances of class Class reside on the heap. Data for loaded types resides in the method area.

# Responsibility of the Class Loader

- locating and importing the binary data for classes
- verify the correctness of imported classes
- allocate and initialize memory for class variables
- assist in the resolution of symbolic references




# Class Loader 동작 과정

**Loading**

- finding and importing the binary data for a type

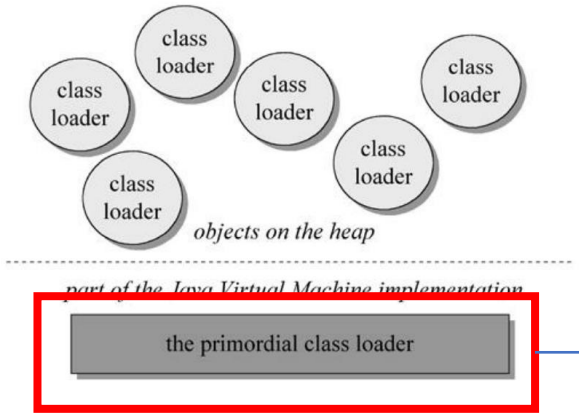
**Linking**

- Verification → ensuring the correctness of the imported type
- Preparation → allocating memory for class variables and initializing the memory to default values
- Resolution → transforming symbolic references from the type into direct references.

**Initialization**

- invoking Java code that initializes class variables to their proper starting values

# The Primordial Class Loader




Every Java Virtual Machine implementation has a primordial class loader, which knows **how to load trusted classes, including the classes of the Java API**

# The Primordial Class Loader 동작 예시

이러한 Class Loader의 동작은 JVM내부와 연결 되어있다.

1. Given a fully qualified type name
2. File 이름을 가지고 CLASSPATH부터 찾는다.
3. 만약에 . Class file이 존재하지 않으면, 하위 디렉토리도 찾는다.
   이때, 패키지 명을 통해 적절한 하위 디렉터리를 찾아갈 수 있다.

이는 하나의 동작 예시이고, 구현마다 달라질 수 있다.

# Namespace by Class Loader

1 Class Loader A

A가 Load한 Class

Class **Cat**
Class Bear

Class Cat 의 Constant Pool

#14 Class = #22 //Dog
...
#22 Utf8 = Dog

2 Class Cat 이 Class Dog를 사용

*Ex) Dog.staticmethodA( )*

To do this,

1. JVM안에 Dog Class에 대한 정보가 있어야 한다.
2. Cat이 Dog Class 정보를 알아야 한다.

3 따라서, Class Loader는

1. Dog라는 이름의 Class file을 JVM내부로 넣어주고
2. 주소 값을 Cat 한테 알려주어야 한다.

**이때, JVM은 Cat Class를 Load한 Class Loader A에게 위 동작을 수행시킨다.**

# Namespace by Class Loader

4 Class Loader A는 위 작업을 수행하고

Class Cat의 Constant Pool의 Dog 값을
실제 Dog Class의 주소로 변환 시킨다.

이를 **Symbol Resolution**이라고 한다.

Class Cat 의 Constant Pool

#14 Class = **#22** //Dog
…
#22 Utf8 = Dog

Class Cat 의 Constant Pool

#14 Class = **실제 Dog Class 주소**
…
#22 Utf8 = Dog

5

이 과정을 생각해보면 Cat은 결국 Class Loader A가 Load할 수 있는 Class들만 참조 할 수 있다는 것을 알 수 있다.

**따라서, 서로 다른 Class Loader에 의해 등록되는 클래스는 서로 간에 완전히 분리되어 있다.**

왜냐하면, Cat Class가 Dog를 참조할 때, 결국 Class Loader A가 Load할 수 없는 Class라면 ClassNotFound Err가 발생할 것이다.