Object Oriented Programming with Java

Sandeep Kulange

Metadata

- A data which describes other data / data about data is called metadata.
- **Consider metadata for Interface**
  - What is name of the interface.
  - In which package it is declared.
  - Which is super interface of that interface.
  - Which annotations are used on interface.
  - Which is access modifier of the interface.
  - Which members are declared inside interface.

Metadata

- **Consider metadata for Class**
  - What is name of the class?
  - In which package it is declared?
  - Which is super class of it?
  - Which interfaces it has implemented?
  - Which is access modifier of it?
  - Whether it is abstract/final?
  - Which annotations are used on it?
  - Which members are declared inside class?

Metadata

- **Consider metadata for Field**
  - What is name of the field?
  - In which class it is declared?
  - Which is access modifier of the field?
  - Which modifiers used on field(static/final/transient)?
  - Which is type of field?
  - Whether it is inherited/declared-only field?
  - Which annotations are used on fields?

Metadata

- **Consider metadata for Method**
  - What is name of the method?
  - In Which class it is declared?
  - Whether it is inherited/declared-only/overridden method?
  - Which is access modifier of the method?
  - Which modifiers used on field(static/final/abstract/synchronized)?
  - Which is return type of the method?
  - What is parameter information of the method?
  - Which exceptions method can throw?
  - Which annotations are used on method?

Application Of Metadata

1. Metadata removes the need for native C/C++ header and library files when compiling because .class file contains bytecode and metadata(information about types defined inside same file and types referenced from other file)
2. IDE uses metadata to help you write code. Its IntelliSense feature parses metadata to tell you what methods, fields, nested types a class contain.
3. Metadata allows an object's fields to be serialized into a memory block, sent to another machine, and then deserialized, re-creating the object's state on the remote machine.
4. Metadata allows the garbage collector to track the lifetime of objects.

Reflection

- It is a Java language feature which provides types(interfaces / classes) that we can use:
    1. To explore metadata
    2. To access private fields of the class
    3. To manage behavior of the application at runtime.

- To use reflection we should use types declared in following package :
    1. java.lang
    2. java.lang.reflect

- Reference:
    1. https://www.oracle.com/technical-resources/articles/java/javareflection.html
    2. https://www.baeldung.com/java-reflection
    3. http://tutorials.jenkov.com/java-reflection/index.html

Reflection

- The java.lang.reflect package provides classes and interfaces for obtaining reflective information about classes and objects.

- Reflection allows programmatic access to information about the fields, methods and constructors of loaded classes, and the use of reflected fields, methods, and constructors to operate on their underlying counterparts, within security restrictions.

- Classes in this package, along with java.lang.Class accommodate applications such as debuggers, interpreters, object inspectors, class browsers, and services such as Object Serialization and JavaBeans that need access to either the public members of a target object (based on its runtime class) or the members declared by a given class.

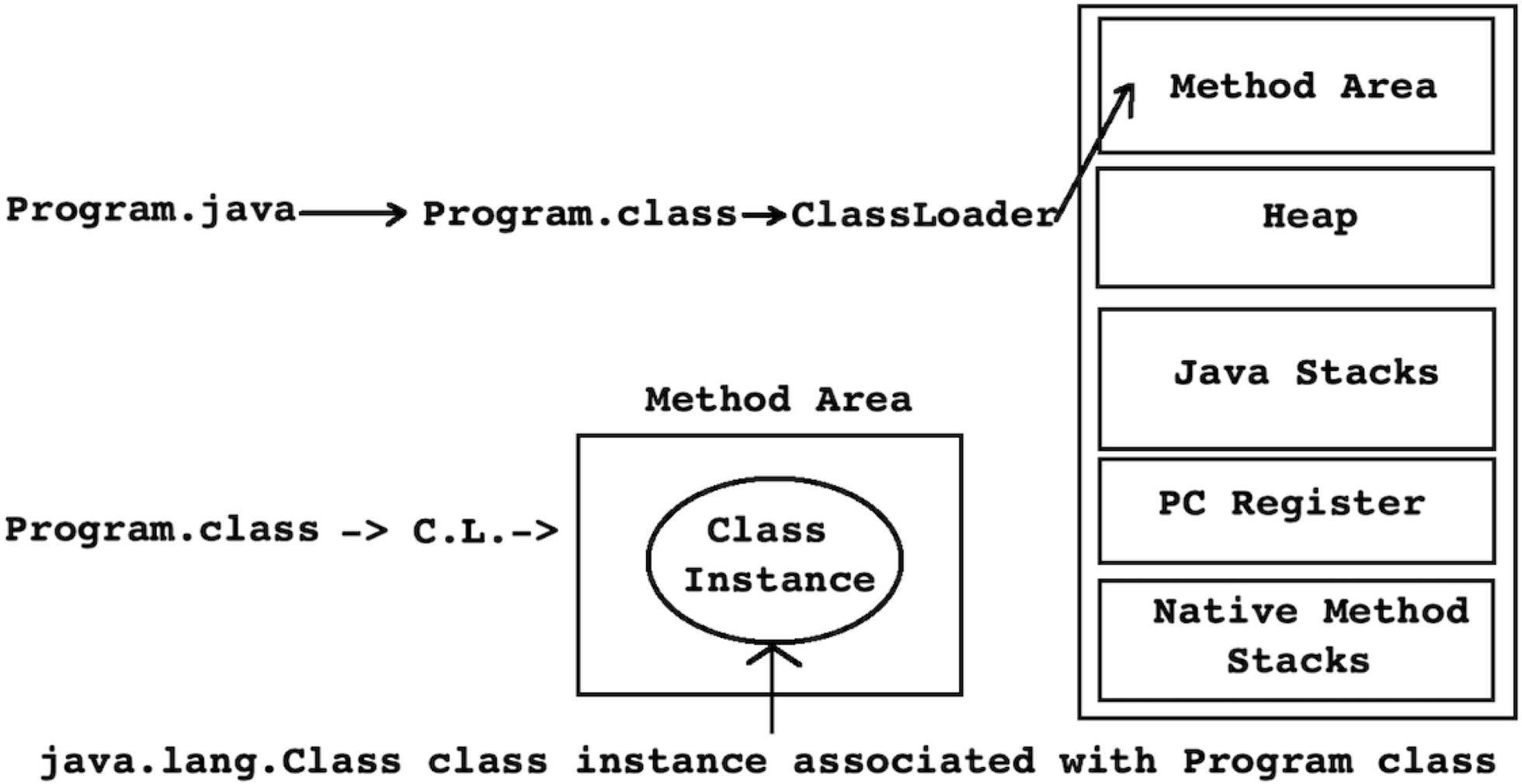
Reflection API

- Types declared in java.lang.reflect package:
  1. AccessibleObject
  2. Array
  3. Constructor
  4. Executable
  5. Field
  6. Method
  7. Modifier
  8. Parameter
  9. Proxy
  10. ReflectPermission
- Type declared in java.lang package:
  1. Class<T>

Class<T> class

- Class<T> is a final class declared in java.lang package.
- Class has no public constructor. Instead Class objects are constructed automatically by the Java Virtual Machine.
- Instances of the class Class represent classes and interfaces in a running Java application. In other words, instance of java.lang.Class class contains metadata of loaded class.
- Methods of java.lang.Class<T>
    1. public static Class<?> forName(String className) throws ClassNotFoundException
    2. public T newInstance()
    3. public Package getPackage()
    4. public String getName()
    5. public String getSimpleName()

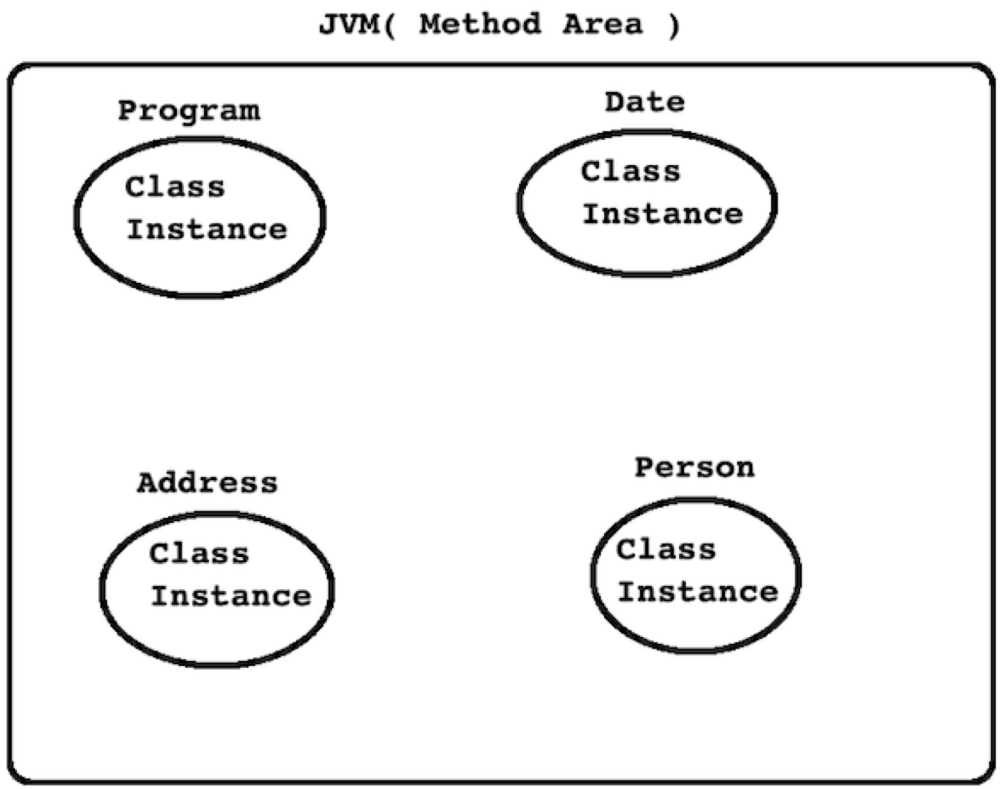
Flow Of Execution



Flow Of Execution

```
class Date{}  //Date.class
class Address{} //Address.class
class Person{ //Person.class
  String name;
  Date dob;
  Address address;
}
class Program{ //Program.class
  p.s.v.m( String[] args ){
    case 1: Date
    case 2: Address
    case 3: Person
  }
}
```

```
javac Program.java
Java Program( Press enter key ) - ClassLoader -
```



Retrieving Class Objects

- The entry point for all reflection operations is java.lang.Class.
- How to get reference of Class<T> class instance associated with loaded type?

1. Using getClass() method
   - "public Class<?> getClass( )" is a method of java.lang.Object class.

```
Integer number = new Integer( 0 );
Class<?> c = number.getClass();
```

2. Using .class syntax

```
Class<?> c = Number.class;
Class<?> c = Integer.class;
```

3. Using Class.forName( )
   - public static Class<?> forName(String className) is a method of Class<T>

```
public static void main(String[] args) {
    try( Scanner sc = new Scanner(System.in)){
     System.out.print("F.Q Name  :   ");
     String className = sc.nextLine();   //java.io.File
     Class<?> c = Class.forName(className);
     } catch (ClassNotFoundException e) {
         e.printStackTrace();
    }
}
```

Thank You.

[sandeepkulange@sunbeaminfo.com]