Java Reflection

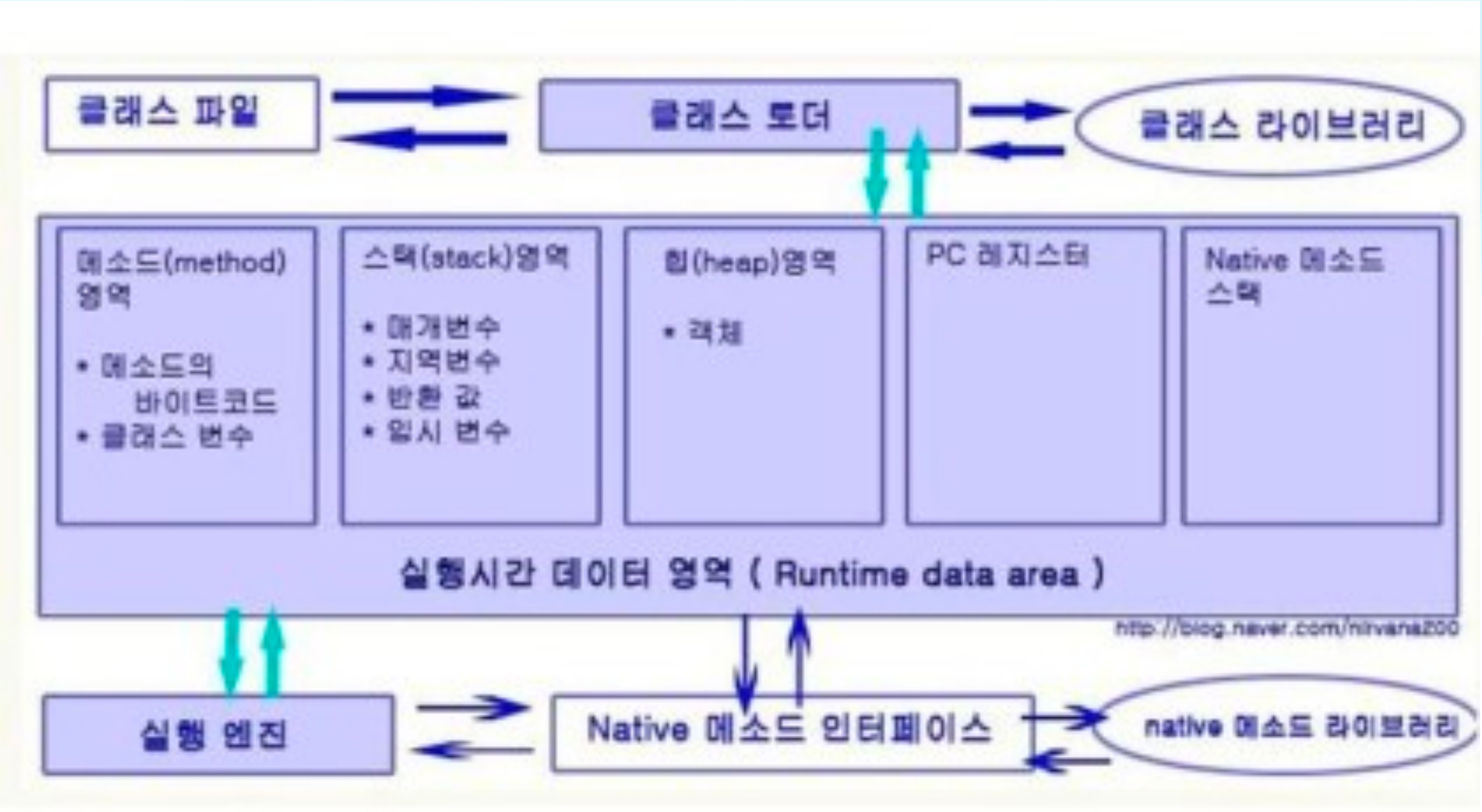
스프링 프레임워크 만들기

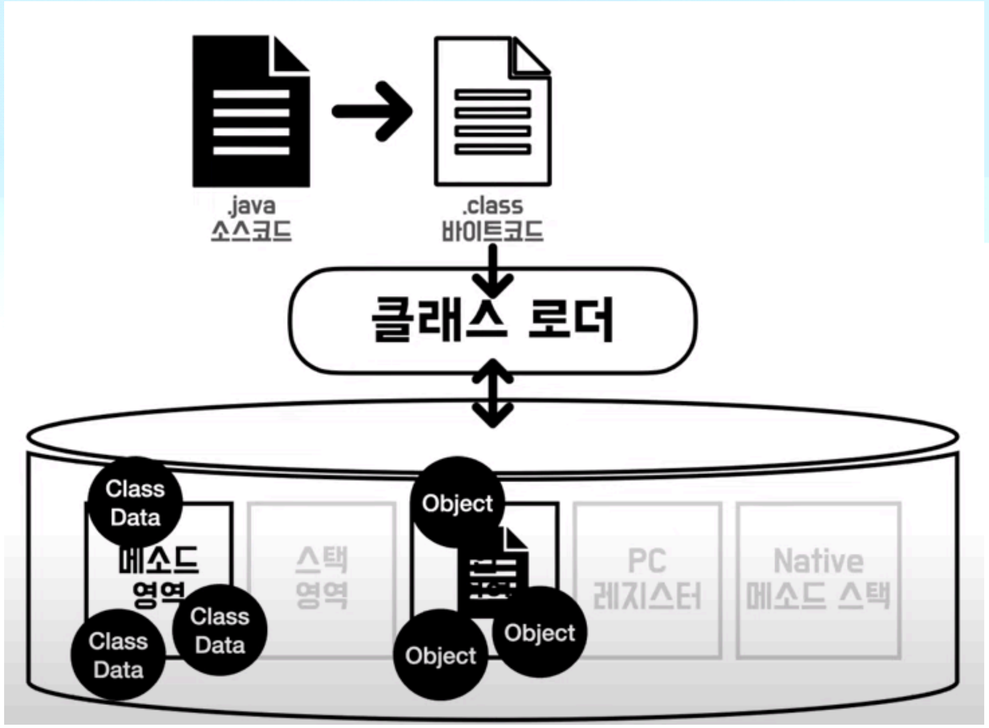
Java Reflection

- 런타임 단계에서 클래스 정보를 분석하거나 수정 할 수 있는 Java API
- 스프링과 각종 프레임워크의 기반이 된다

JVM의 작동 원리

동적 클래스 로딩





지정된 API를 활용해서만 class 객체 사용

```java
new *
public class Main {
    new *
    public static void main(String[] args) throws InterruptedException, ClassNotFoundException {
        // 1
        Class<?> aClass = Class.forName( className: "org.example.Student");
        // 2
        Class<String> stringClass = String.class;
        // 3
        Class<? extends String[]> aClass1 = args.getClass();
    }
}
```

```java
@NotNull
@CallerSensitive
public static Class<?> forName( @NonNls String className)
            throws ClassNotFoundException {
    Class<?> caller = Reflection.getCallerClass();
    return forName(className, caller);
}

// Caller-sensitive adapter method for reflective invocation
@CallerSensitiveAdapter
private static Class<?> forName(String className, Class<?> caller)
        throws ClassNotFoundException {
    ClassLoader loader = (caller == null) ? ClassLoader.getSystemClassLoader()
                                          : ClassLoader.getClassLoader(caller);
    return forName0(className, initialize: true, loader, caller);
}
```

> Called after security check for system loader access checks have been made.

```java
private static native Class<?> forName0(String name, boolean initialize,
                                        ClassLoader loader,
                                        Class<?> caller)
    throws ClassNotFoundException;
```

Class 객체

우리가 만들 수는 없다.

```
public final class Class<T> implements java.io.Serializable,
                              GenericDeclaration,
                              Type,
                              AnnotatedElement,
                              TypeDescriptor.OfField<Class<?>>,
                              Constable {
    private static final int ANNOTATION= 0x00002000;
    private static final int ENUM      = 0x00004000;
    private static final int SYNTHETIC = 0x00001000;

    private static native void registerNatives();
    static {
        registerNatives();
    }

    /*
     * Private constructor. Only the Java Virtual Machine creates Class objects.
     * This constructor is not used and prevents the default constructor being
     * generated.
     */
    private Class(ClassLoader loader, Class<?> arrayComponentType) {
        // Initialize final field for classLoader.  The initialization value of non-null
        // prevents future JIT optimizations from assuming this final field is null.
        classLoader = loader;
        componentType = arrayComponentType;
    }
```

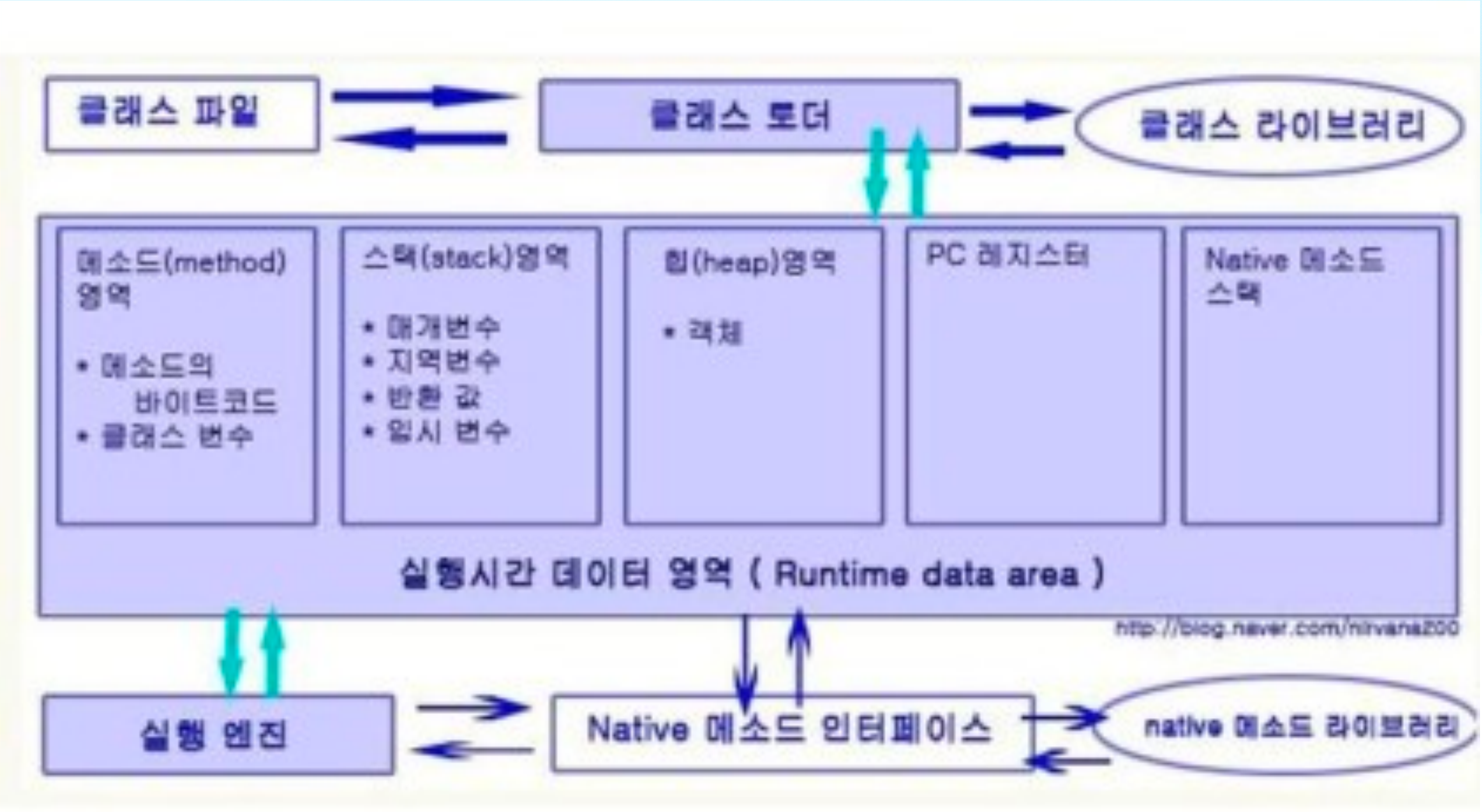
```
@NotNull
@CallerSensitive
public static Class<?> forName( @NonNls String className)
            throws ClassNotFoundException {
    Class<?> caller = Reflection.getCallerClass();
    return forName(className, caller);
}

// Caller-sensitive adapter method for reflective invocation
@CallerSensitiveAdapter
private static Class<?> forName(String className, Class<?> caller)
        throws ClassNotFoundException {
    ClassLoader loader = (caller == null) ? ClassLoader.getSystemClassLoader()
                                          : ClassLoader.getClassLoader(caller);
    return forName0(className, initialize: true, loader, caller);
}
```

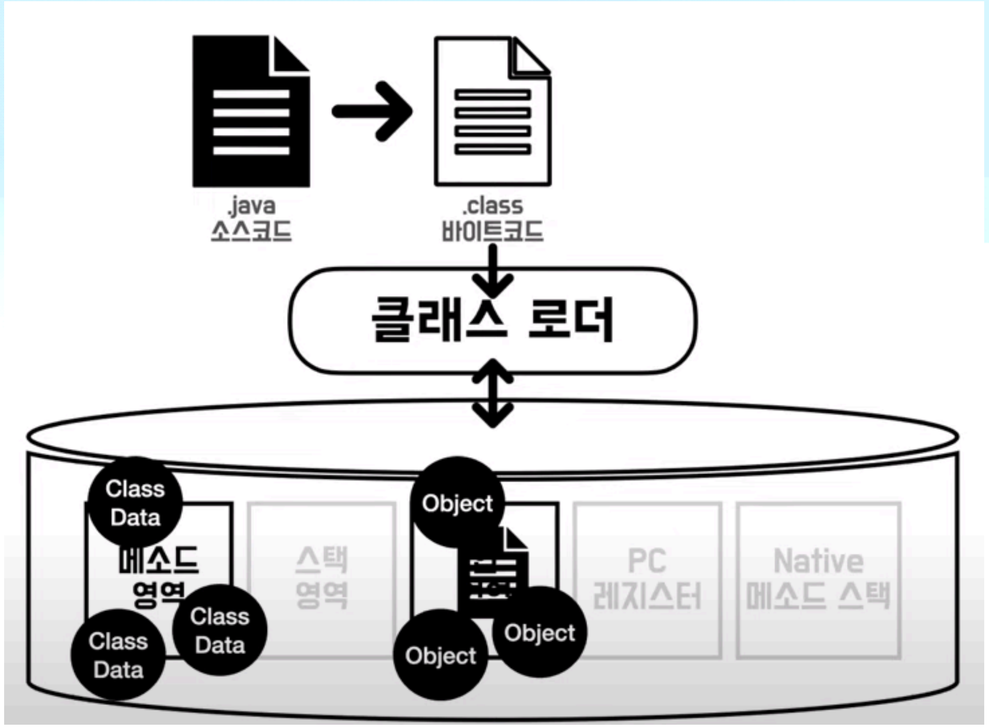
> Called after security check for system loader access checks have been made.

```
private static native Class<?> forName0(String name, boolean initialize,
                                        ClassLoader loader,
                                        Class<?> caller)
    throws ClassNotFoundException;
```

JVM의 작동 원리

동적 클래스 로딩





사용 예시를 보며 익히기

테스트를 위한 클래스 정의

```java
@SslBean
public class Student {

    4 usages
    private String name;
    no usages
    private String hiddenName;

    //생성자
    no usages  new *
    public Student() {
    }
    no usages  new *
    public Student(String name) {
        this.name = name;
    }

    //getter, setter
    new *
    public String getName() {
        return name;
    }
    no usages  new *
    public void setName(String name) {
        this.name = name;
    }

    no usages  new *
    @SslMethod
    public String sayHello() {
        return "Hello " + name;
    }
    no usages  new *
    private String hiddenMethod() {
        return "hidden";
    }
```

```java
1 usage  new *
@Target({ElementType.TYPE})
@Retention(java.lang.annotation.RetentionPolicy.RUNTIME)
public @interface SslBean {
}
```

```java
1 usage  new *
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface SslMethod {
}
```

생성해서 public getter 호출

```
public static void main(String[] args) throws InterruptedException, ClassNotFoundExcep
    // 클래스 불러오기
    Class<?> aClass = Class.forName( className: "org.example.Student");

    // 클래스 인스턴스 생성
    Constructor<?> constructor = aClass.getConstructor(String.class);
    Student student = (Student) constructor.newInstance( ...initargs: "홍길동");

    // 리플렉션을 이용해서 getName 메소드 호출
    String getName = (String) aClass.getMethod( name: "getName").invoke(student);
    System.out.println(getName);

    //출력 : 홍길동

}
```

특정 어노테이션이 붙어있는 클래스 정보

```java
public static void main(String[] args) throws InterruptedException, ClassNotFoundException, InstantiationException, Illega
    // 클래스 불러오기
    Class<?> aClass = Class.forName( className: "org.example.Student");

    // 클래스 인스턴스 생성
    Constructor<?> constructor = aClass.getConstructor(String.class);
    Student student = (Student) constructor.newInstance( ...initargs: "홍길동");

    // SslBean 어노테이션 확인
    SslBean sslBean = aClass.getAnnotation(SslBean.class);
    System.out.println(sslBean != null ? "@SslBean이 붙은 클래스 입니다." : "@SslBean이 붙지 않은 클래스 입니다.");

    // SslMethod 가 붙어있는 메소드 이름 확인
    Method[] methods = aClass.getDeclaredMethods();
    for (Method method : methods) {
        SslMethod sslMethod = method.getAnnotation(SslMethod.class);
        if (sslMethod != null) {
            // 메소드 이름 출력
            System.out.println("@SslMethod 가 붙은 메서드 이름은 : " +  method.getName());

            // 메소드 실행
            Object invoke = method.invoke(student);
            System.out.println("실행결과는 : " + invoke);
        }
    }
    /*
    출력 :
    @SslBean이 붙은 클래스 입니다.
    @SslMethod 가 붙은 메서드 이름은 : sayHello
    실행결과는 : Hello 홍길동
     */
```

Private 강제 접근

테스트 코드 작성시 유용함

```java
public static void main(String[] args) throws InterruptedException, ClassNotFoundException, InstantiationExcep
    // 클래스 불러오기
    Class<?> aClass = Class.forName( className: "org.example.Student");

    // 클래스 인스턴스 생성
    Constructor<?> constructor = aClass.getConstructor(String.class);
    Student student = (Student) constructor.newInstance( ...initargs: "홍길동");

    // private 필드에 직접 값 넣기
    Field hiddenField = aClass.getDeclaredField( name: "hiddenName");
    hiddenField.setAccessible(true);
    hiddenField.set(student, "이순신");

    // private 메소드 호출
    Method hiddenMethod = aClass.getDeclaredMethod( name: "hiddenMethod");
    hiddenMethod.setAccessible(true);
    String result = (String) hiddenMethod.invoke(student);
    System.out.println(result);
    // 결과: hidden

}
```

```java
@SslBean
public class Student {

    4 usages
    private String name;
    no usages
    private String hiddenName;

    //생성자
    no usages  new *
    public Student() {
    }
    no usages  new *
    public Student(String name) {
        this.name = name;
    }

    //getter, setter
    new *
    public String getName() {
        return name;
    }
    no usages  new *
    public void setName(String name) {
        this.name = name;
    }

    no usages  new *
    @SslMethod
    public String sayHello() {
        return "Hello " + name;
    }
    no usages  new *
    private String hiddenMethod() {
        return "hidden";
    }
```

나만의 Spring Framework 만들어보기

```java
//내가 만든 스프링
5 usages  new *
public class CustomApplicationContext {
    3 usages
    private ConcurrentHashMap<String, Object> singletonBeans = new ConcurrentHashMap<>();

    1 usage  new *
    public Object getBean(String name) {
        return singletonBeans.get(name);
    }

    1 usage  new *
    public <T> T getBean(Class<T> clazz) {
        return singletonBeans.entrySet().stream() Stream<Entry<...>>
                .filter(entry -> clazz.isAssignableFrom(entry.getValue().getClass()))
                .map(entry -> entry.getValue()) Stream<Object>
                .findFirst() Optional<Object>
                .map(bean -> (T) bean) Optional<T>
                .orElseThrow(() -> new RuntimeException("No bean found"));
    }

    1 usage  new *
    public static CustomApplicationContext run(String basePackage) throws IOException {
        CustomApplicationContext context = new CustomApplicationContext();
        context.init(basePackage);

        return context;
    }
```

```java
1 usage  new *
private void init(String packageName) throws IOException {
    //클래스 로더로 불러오기
    InputStream stream = ClassLoader.getSystemClassLoader()
            .getResourceAsStream(packageName.replaceAll( regex: "[.]",  replacement: "/"));
    BufferedReader reader = new BufferedReader(new InputStreamReader(stream));
    reader.lines() Stream<String>
            .filter(line -> line.endsWith(".class"))
            .map(line -> getClass(line, packageName)) Stream<Class>
            .filter(Objects::nonNull)
            //SslBean 어노테이션이 붙은 클래스만 필터링
            .filter(clazz -> clazz.isAnnotationPresent(SslBean.class))
            .forEach(clazz -> {
                try {
                    //객체를 생성해서, 싱글톤 빈에 등록
                    Constructor<?> constructor = clazz.getConstructor();
                    Object instance = constructor.newInstance();
                    singletonBeans.put(clazz.getSimpleName(), instance);
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
            });
    reader.close();
}
```

잘 돌아가나?

의존성 주입도 구현가능!

```
new *
public static void main(String[] args) throws IOException {
    //내가 만든 스프링 실행
    final CustomApplicationContext context;
    context = CustomApplicationContext.run( basePackage: "org.example");

    //이름으로 빈 주입 받기
    Student beanByName = (Student) context.getBean( name: "Student");
    System.out.println("beanByName.getName : " + beanByName.getName());
    System.out.println("beanByName.sayHello : " + beanByName.sayHello());

    System.out.println();
    //타입으로 빈 주입 받기
    Student beanByType = context.getBean(Student.class);
    System.out.println("beanByType.getName : " + beanByType.getName());
    System.out.println("beanByType.sayHello : " + beanByType.sayHello());

    System.out.println();

    //둘이 같은지 (싱글톤인지 확인)
    System.out.println("beanByName == beanByType : " + (beanByName == beanByType));
}
/*
beanByName.getName : default
beanByName.sayHello : Hello default

beanByType.getName : default
beanByType.sayHello : Hello default

beanByName == beanByType : true
 */
```

감사합니다