REFLECTION API

BARIŞ DEMİRCİ

REFLECTION API

- Java 1 ile beraber kullanıma sunulmuştur.
- Method Handles kullanımı Java 7 ile gelmiştir.
- Reflection API, javada bir class'taki methodlara ve fieldlara erişebildiğimiz bir API olarak karşımıza çıkar.
- Nesnenin hangi classtan olduğunu veya yapısını bilmeye gerek kalmadan; içeriğini okumaya ve değiştirmeye yardımcı olur.
- Java Reflection API, Java programlarının çalışma zamanında sınıf yapısını ve nesne davranışını incelemesine ve değiştirmesine olanak tanıyan güçlü bir özelliktir.

NEDEN REFLECTION API ÖNEMLİ?

- **1. Dinamik Yükleme ve Çalıştırma**

Reflection, programların çalışma zamanında sınıfları, metodları ve değişkenleri dinamik olarak yükleyip çalıştırmasına izin verir. Bu, uygulamanın daha esnek ve genişletilebilir olmasını sağlar.

NEDEN REFLECTION API ÖNEMLİ?

- **2. Çerçeve ve Kütüphane Geliştirme:**

Reflection, birçok Java çerçevesi ve kütüphanesi tarafından kullanılır. Örneğin, Spring ve Hibernate gibi popüler çerçeveler, çalışma zamanında bağımlılık enjeksiyonu ve ORM işlemlerini gerçekleştirmek için Reflection API'yi kullanır.

NEDEN REFLECTION API ÖNEMLİ?

- **3. Kapsamlı Bilgi Erişimi:**

Reflection, bir sınıfın yapısını (örneğin, metodlar, alanlar, constructor'lar) çalışma zamanında inceleyebilme yeteneği sağlar. Bu, özellikle genel (generic) tiplerle çalışma veya başka bir şekilde derleme zamanında bilinmeyen tiplerle etkileşimde bulunma durumunda faydalıdır.

NEDEN REFLECTION API ÖNEMLİ?

- **4. Kod Üretme ve Test Etme:**

Reflection, kod üretme araçları ve test çerçevelerinde yaygın olarak kullanılır. JUnit gibi test çerçeveleri, test metotlarını otomatik olarak bulmak ve çalıştırmak için Reflection kullanır. Bu, test yazımını ve bakımını kolaylaştırır.

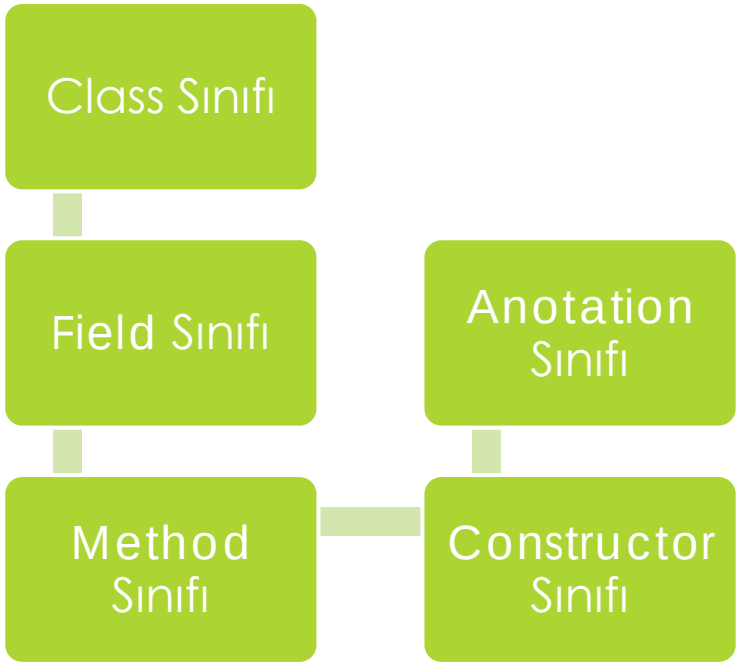
NEDEN REFLECTION API ÖNEMLİ?

- **5. Bağımlılık Enjeksiyonu ve Nesne Yönetimi:**

Dependency Injection (DI) ve Inversion of Control (IoC) kalıpları, çalışma zamanında nesnelerin bağımlılıklarını enjekte etmek için Reflection kullanır. Bu, uygulamaların daha gevşek bağlı olmasını sağlar ve modülerlik ile test edilebilirlik sağlar.

REFLECTION API

- Bu api içinde önemli bazı sınıflar vardır. Bu sınıflar sayesinde class'ın her bir öğesine erişim için modeller sunulmuş olur.



Class sınıfı

- Class sınıfından bir **instance üretemeyiz.**
- Bir nesnenin sınıfını sorgulayarak Class sınıfına aktarabiliriz.
- Sınıfı derleme zamanındaki adıyla alabiliriz.
- Ayrıca daha da güzeli **runtime'daki** adıyla da alabiliriz.

- Objenin sınıfını almanın yolları vardır. Bunları uygulamada görelim.

.getClass() metodu

- Java'daki tüm nesneler, java.lang.Object sınıfından getClass() metodunu miras alır. Bu metod, örneğin çalışma zamanındaki sınıfını temsil eden bir Class nesnesi döner.
- Bir nesne örneğiniz olduğunda bu yöntem kolay ve doğrudan bir yaklaşımdır.

```
String s = "Hello, Reflection!";
Class<?> clazz = s.getClass();
System.out.println(clazz.getName()); // "java.lang.String" yazdırır
```

.class metodu

- Bazen, bir nesne örneklemeden Class nesnesini elde etmek isteyebilirsiniz. Bu gibi durumlarda, sınıf isimleri veya ilkel veri tipleri üzerinde .class sözdizimini kullanmak doğrudan bir çözüm sunar.

```
Class<?> intClass = int.class;
Class<?> listClass = java.util.ArrayList.class;
System.out.println(intClass.getName()); // "int" yazdırır
System.out.println(listClass.getName()); // "java.util.ArrayList" yazdırır
```

✓ Bu yaklaşım, özellikle ilkel veri tipleriyle çalışırken kullanışlıdır, çünkü bunlar normal nesneler gibi örneklenemezler.

Class.forName(String className) Kullanımı

- Reflection araçlarınız arasında güçlü bir yöntem olan Class.forName() metodudur. Bu metod, bir sınıfı tam nitelikli adıyla dinamik olarak yüklemenizi ve bağlamanızı sağlar.

```
try {
    Class<?> dynamicClass = Class.forName("java.util.HashMap");
    System.out.println(dynamicClass.getSimpleName()); // "HashMap" yazdırır
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

- ✓ Class.forName() metodu sadece Class nesnesini döndürmekle kalmaz, aynı zamanda sınıfın statik başlatılmasını da gerçekleştirir. Bu davranış, sürücülerin bağlantı dizesine dayalı olarak dinamik olarak yüklendiği JDBC (Java Database Connectivity) uygulamalarında çok önemlidir.

Reflection Api ile Constructorlara Erişim

- Reflection API kullanarak bir sınıfın constructorlarına erişebilir ve bu constructorları kullanarak nesne oluşturabilirsiniz.

- Class sınıfının
  - getConstructors() metodu ile public constructorlara,
  - getDeclaredConstructors() metodu ile tüm constructorlara erişebilirsiniz.

- Constructor.newInstance(Object... initargs) metodu ile constructor üzerinden nesne oluşturabilirsiniz.

- Private constructorlara erişim setAccessible(true) metodu ile sağlanabilir.

Reflection Api ile Metodlara Erişim

- Reflection API, çalışma zamanında sınıfın metodlarına erişim ve bu metodlar üzerinde işlem yapma olanağı tanır.
- `Class` sınıfının
  - `getMethods()` metodu ile public metodlara,
  - `getDeclaredMethods()` metodu ile tüm metodlara erişebilirsiniz.
- Metodların adlarını, parametre türlerini, dönüş türlerini ve diğer özelliklerini inceleyebilirsiniz.
- `Method.invoke(Object obj, Object... args)` metodu ile belirli bir nesne üzerinde belirli bir metodu çalıştırabilirsiniz.
- Private metodlara erişim `setAccessible(true)` metodu ile sağlanabilir.
- Metodların parametre türleri ve dönüş değerleri `Method.getParameterTypes()` ve `Method.getReturnType()` metodları ile incelenebilir.
- Metodlar üzerinde tanımlı annotationlar `Method.getAnnotations()` metodu ile alınabilir ve işlenebilir.

Reflection Api ile Fieldlara Erişim

- Reflection API, çalışma zamanında bir sınıfın alanlarına (fields) erişim ve bu alanlar üzerinde işlem yapma olanağı tanır.
- Class sınıfının
  - getFields() metodu ile public alanlara,
  - getDeclaredFields() metodu ile tüm alanlara erişebilirsiniz.
- Alanların adlarını, türlerini ve diğer özelliklerini inceleyebilirsiniz.
- Field.get(Object obj) metodu ile belirli bir nesnenin alan değerini alabilir, Field.set(Object obj, Object value)metodu ile belirli bir nesnenin alanına değer atayabilirsiniz.
- Private alanlara erişim setAccessible(true) metodu ile sağlanabilir.
- Alanların türleri Field.getType() metodu ile, erişim belirleyicileri (modifiers) Field.getModifiers() metodu ile alınabilir.
- Alanlar üzerinde tanımlı annotationlar Field.getAnnotations() metodu ile alınabilir ve işlenebilir.

Reflection Api ile Super Class'a Erişim

- Reflection API, bir sınıfın SuperClass'ına erişim ve bu SuperClass üzerinde işlem yapma olanağı sağlar.
- Class sınıfının
  - getSuperclass() metodu ile bir sınıfın SuperClass'ını elde edebilirsiniz.
- SuperClass'ın adını, metotlarını, alanlarını ve diğer özelliklerini inceleyebilirsiniz.
- SuperClass'a erişim, belirli bir sınıfın kalıtım hiyerarşisini daha iyi anlamanızı sağlar ancak bu işlem bazı durumlarda yavaşlatıcı olabilir.
- Reflection API kullanırken, kalıtım yapısını ve sınıflar arasındaki ilişkileri anlamak önemlidir.

Reflection Api ile Interface'lere Erişim

- Reflection API, bir sınıfın implement ettiği interfacelere erişim ve bu interfaceler üzerinde işlem yapma olanağı sağlar.
- Class sınıfının
  - getInterfaces() metodu ile bir sınıfın implement ettiği interfacelerin listesini alabilirsiniz.
- Interface'lerin adlarını, metotlarını, sabitlerini ve diğer özelliklerini inceleyebilirsiniz.
- Reflection API kullanarak interfacelere erişim, bir sınıfın nasıl kullanılacağını daha iyi anlamanıza ve dinamik davranışlar eklemenize olanak tanır.Interface'ler genellikle bir API'nin kullanımını tanımlar ve bu API'ler üzerinde çalışma zamanında dinamik değişiklikler yapabilirsiniz.
- Java'da interface'ler, polimorfizmi destekler ve bir nesnenin hangi sınıf veya interfaceleri implement ettiğini reflection ile belirleyebilirsiniz.

Reflection Api ile Class Modifier Erişim

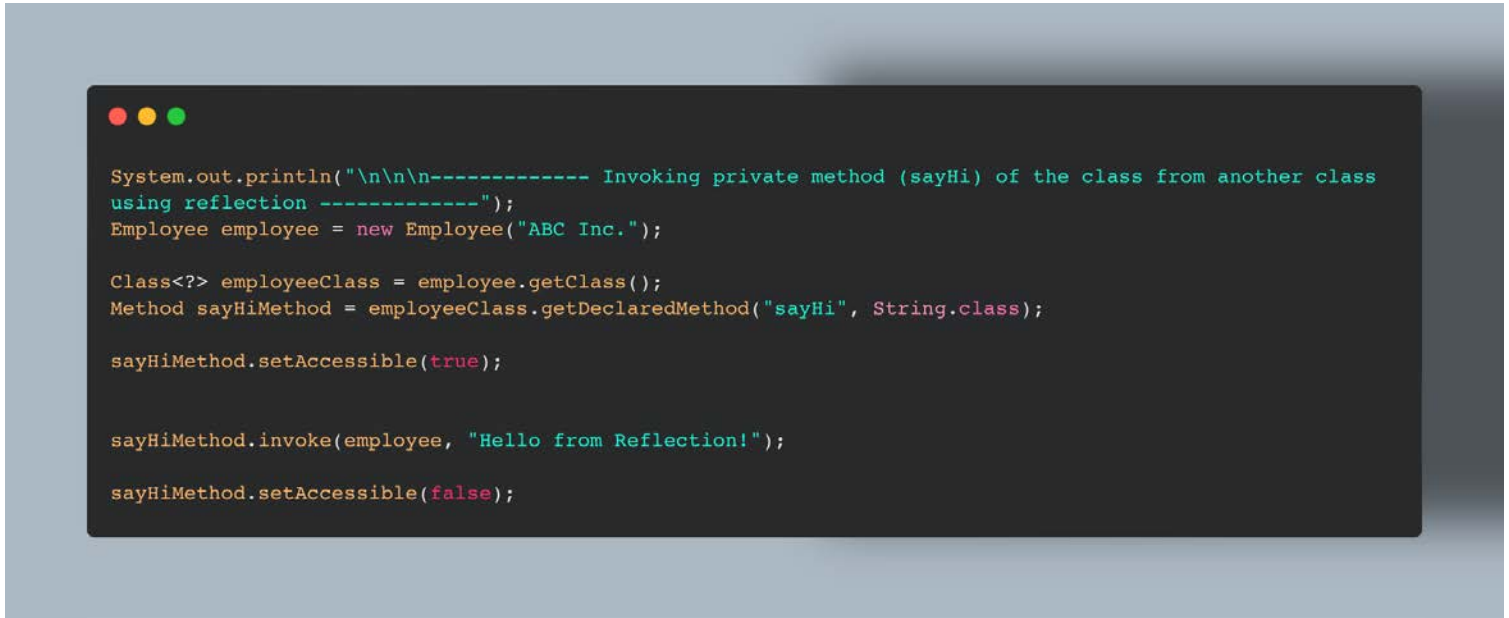
- Reflection API, bir sınıfın erişim belirleyicilerini (modifiers) ve diğer özelliklerini inceleme ve bu özelliklere erişim sağlama imkanı sunar.
- Class sınıfının
  - getModifiers() metodu ile bir sınıfın erişim belirleyicilerini (public, private, protected vb.) ve diğer modifierları (final, abstract vb.) alabilirsiniz.
- Modifier'ların değerlerini, isimlerini ve anlamlarını inceleyebilirsiniz.
- Modifier'lar sınıfların ve interfacelerin kullanımını ve davranışını belirler. Bu nedenle, sınıf veya interface üzerinde yapılacak değişiklikler dikkatli planlanmalıdır.
- Java dilinde tanımlı Modifier sabitleri, sınıfların ve interfacelerin yapısal özelliklerini tanımlamak için kullanılır ve Reflection API ile erişilebilir.

Reflection Api ile Anotasyonlara Erişim

- Reflection API, bir sınıfın, metotun veya alanın üzerinde tanımlı olan anotasyonlara erişim ve bu anotasyonlar üzerinde işlem yapma olanağı sağlar.
- Class, Method, Field gibi sınıfların getAnnotations() veya getDeclaredAnnotations() metotları ile tanımlı anotasyonları alabilirsiniz.
- getAnnotation(Class<T> annotationClass) metodu ile belirli bir anotasyonu alabilirsiniz.
- Anotasyonların adını, parametrelerini ve diğer özelliklerini inceleyebilirsiniz.
- Anotasyonlar, Java'da metadata olarak bilgiler sunar ve sınıfların veya metodların davranışını veya yapısal özelliklerini tanımlar.
- Reflection API ile anotasyonlara erişim, kodunuzun daha esnek ve dinamik olmasını sağlar ancak anotasyonların doğru ve amaca uygun kullanılması önemlidir.

Reflection Api ile private Methodları Invoke etme

- Reflection API kullanarak private bir metodu tetiklemek, Java'da güvenlik ve erişim kısıtlamaları nedeniyle doğrudan yapılamaz. Ancak, bazı durumlarda bu kısıtlamaları aşmanın yolları vardır:
- **setAccessible(true) Metodu Kullanma:**
  : java.lang.reflect.Method sınıfının setAccessible(true) metodu, private metodlara erişimi sağlar.



```
System.out.println("\n\n\n------------- Invoking private method (sayHi) of the class from another class
using reflection -------------");
Employee employee = new Employee("ABC Inc.");

Class<?> employeeClass = employee.getClass();
Method sayHiMethod = employeeClass.getDeclaredMethod("sayHi", String.class);

sayHiMethod.setAccessible(true);


sayHiMethod.invoke(employee, "Hello from Reflection!");

sayHiMethod.setAccessible(false);
```

Reflection Api ile private Fieldlara erişim

- Reflection API kullanarak private field'lara erişim, Java'da bazı güvenlik kısıtlamaları nedeniyle doğrudan yapılamaz.
  Ancak, setAccessible(true) metodu kullanılarak bu kısıtlamalar aşılabilir.

```java
import java.lang.reflect.Field;

public class ReflectionPrivateFieldExample {
    public static void main(String[] args) {
        try {
            // Sınıfı alma
            Class<?> clz = Class.forName("com.barisd.Person");

            // Nesne oluşturma
            Object personInstance = clz.getDeclaredConstructor().newInstance();

            // Private field'a erişim
            Field nameField = clz.getDeclaredField("name");
            nameField.setAccessible(true); // Private field'a erişim izni sağlar

            // Private field'ın değerini okuma
            String nameValue = (String) nameField.get(personInstance);
            System.out.println("Name before change: " + nameValue);

            // Private field'ın değerini değiştirme
            nameField.set(personInstance, "Baris D.");
            String newNameValue = (String) nameField.get(personInstance);
            System.out.println("Name after change: " + newNameValue);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

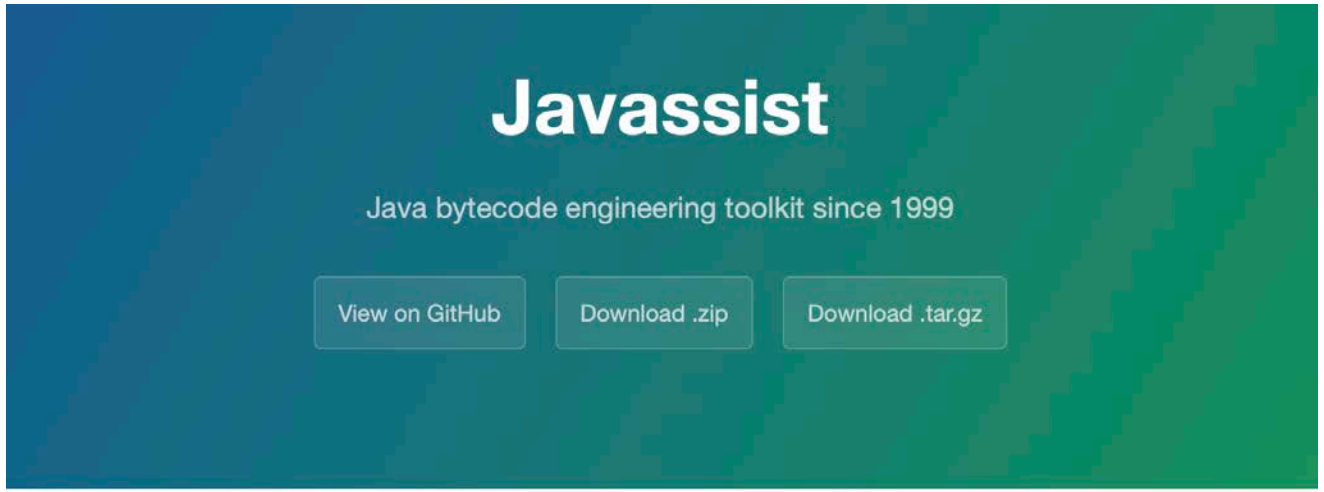
Reflection Api ile Class Instance Oluşturma

- Reflection API kullanarak bir sınıfın nesnesini (instance) dinamik olarak oluşturabilirsiniz. Bu işlem, genellikle bir sınıfın türünü önceden bilmediğiniz veya dinamik olarak belirlemek istediğiniz durumlarda kullanılır.

```java
import java.lang.reflect.Constructor;

public class ReflectionInstanceExample {
    public static void main(String[] args) {
        try {
            // Sınıfı alma
            Class<?> clz = Class.forName("com.barisd.Person");

            // Varsayılan constructor ile instance oluşturma
            Object personInstance1 = clz.getDeclaredConstructor().newInstance();
            System.out.println("Instance 1: " + personInstance1);

            // Parametreli constructor ile instance oluşturma
            Constructor<?> constructor = clz.getDeclaredConstructor(String.class, int.class);
            Object personInstance2 = constructor.newInstance("Baris D.", 38);
            System.out.println("Instance 2: " + personInstance2);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Javassist

- Javassist, Java sınıflarını ve metodlarını çalışma zamanında dinamik olarak değiştirmeyi kolaylaştıran bir bytecode manipulation kütüphanesidir.



Javassist (Java Programming Assistant) makes Java bytecode manipulation simple. It is a class library for editing bytecodes in Java; it enables Java programs to define a new class at runtime and to modify a class file when the JVM loads it. Unlike other similar bytecode editors, Javassist provides two levels of API: source level and bytecode level. If the users use the source-level API, they can edit a class file without knowledge of the specifications of the Java bytecode. The whole API is designed with only the vocabulary of the Java language. You can even specify inserted bytecode in the form of source text; Javassist compiles it on the fly. On the other hand, the bytecode-level API allows the users to directly edit a class file as other editors.

ASM

- ASM, Java sınıflarının bayt kodunu okuma, yazma ve değiştirme işlemlerini gerçekleştirmek için kullanılan düşük seviyeli bir bytecode manipulation kütüphanesidir. ASM, performansı ve esnekliği nedeniyle oldukça popülerdir.