انجمن جاوا کاپ تقدیم می‌کند

دوره برنامه‌نویسی جاوا

بازتاب
Reflection

صادق علی‌اکبری

حقوق مؤلف

- کلیه حقوق این اثر متعلق به انجمن جاواکاپ است
- بازنشر یا تدریس آن‌چه توسط جاواکاپ و به صورت عمومی منتشر شده است، با ذکر مرجع (جاواکاپ) بلامانع است
- اگر این اثر توسط جاواکاپ به صورت عمومی منتشر نشده است و به صورت اختصاصی در اختیار شما یا شرکت شما قرار گرفته، بازنشر آن مجاز نیست
- تغییر محتوای این اثر بدون اطلاع و تأیید انجمن جاواکاپ مجاز نیست

سرفصل مطالب

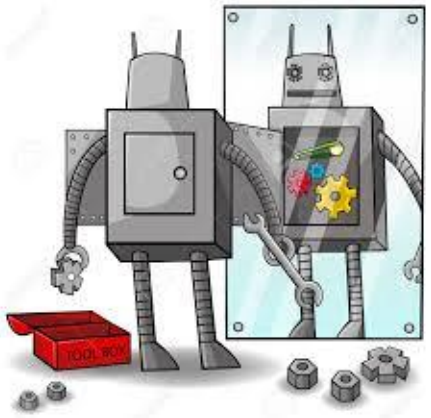
- نیاز به اطلاعات نوع شیء در زمان اجرا
- استفاده از این اطلاعات در زمان اجرا
- امکانات جاوا در این زمینه: بازتاب (Reflection)
- مفهوم بازتاب در برنامه‌نویسی
- بارگذاری پویا (Dynamic Loading)
- شیء کلاس (Class Object)

مقدمه

بازتاب (Reflection) چیست؟

- وقتی به آینه نگاه می‌کنیم: بازتاب ویژگی‌ها و رفتار خودمان را می‌بینیم
- معنای بازتاب (Reflection) در برنامه‌نویسی:
  - برنامه بتواند خودش را ببیند و تغییراتی در خودش اعمال کند
  - کدی که در زمان اجرا، اطلاعات همان برنامه را ببیند، بررسی کند و بتواند تغییر دهد
  - بازتابِ امکاناتِ برنامه که در اجرای همان برنامه قابل استفاده است

    (امکاناتی مثل کلاس‌ها، ویژگی‌ها، متدها و ...)

سؤال

- چگونه متدی بنویسیم که

نام کلاس را به عنوان پارامتر بگیرد و یک شیء جدید از آن کلاس را ایجاد کند و برگرداند؟

- مثلاً چگونه متد instantiate را پیاده‌سازی کنیم:

```
Object o = instantiate("java.lang.String");
String s = (String) o;

Person p = (Person)instantiate("ir.javacup.hr.Person");
```

- و یا چگونه با داشتن نام یک متد (یک رشته) آن متد را روی یک شیء فراخوانی کنیم؟
- نیازهایی مثل موارد فوق، با امکاناتی که تاکنون دیده‌ایم قابل پیاده‌سازی نیست
- این نیازها به «عملیاتی با نوع داده در زمان اجرا» ممکن می‌شود

سؤال (۲)

- چگونه برنامه‌ای بنویسیم که حاشیه‌نویسی‌ها (annotations) را بررسی کند؟
- مثلاً کلاس‌هایی که @WebService دارند را پیدا کند
  - و خدمات خاصی برای اشیاء این کلاس‌ها ارائه کند
- و یا هنگام فراخوانی متدهایی که @Transactional دارند، تمهیداتی بیندیشد
- و این کارها را در زمان اجرای برنامه انجام دهد؟

```
@WebService
class Account{
  @Transactional
  public void transfer(double amount, Account to){
 ...
```

اطلاعاتی درباره نوع اشیاء در زمان اجرا

- در زبان جاوا می‌توانیم در زمان اجرا اطلاعاتی درباره نوع اشیاء را استخراج کنیم
  - و همچنین کارهایی با نوع داده موردنظر انجام دهیم
- مثال:
  - ارجاع موردنظر به چه نوع شیئی اشاره می‌کند؟
  - نوع شیء موردنظر، شامل چه متدها و ویژگی‌هایی است؟
  - یک نمونه جدید از نوع موردنظر ایجاد کنیم
  - متد موردنظر را روی شیء موردنظر فراخوانی کنیم

- امکانات جاوا در این زمینه، بازتاب (Reflection) خوانده می‌شود

برخی از امکانات بازتاب (Reflection)

- نوع دقیق شیء را در زمان اجرا تشخیص دهیم
  - مثلاً نوع شیئی که متغیر x به آن اشاره خواهد کرد، در زمان کامپایل معلوم نیست:

```
Object x = (a>b? new String("A"): new Person("Ali"));
```

- اطلاعاتی درباره امکانات یک کلاس به دست آوریم
  - مثل فهرست متدها، سازنده‌ها، فیلدها و ...
- ایجاد یک شیء با داشتن نام کلاس
- فراخوانی یک متد روی یک شیء با داشتن نام متد

- امکانات جاوا در زمینه بازتاب در بسته‌ی java.lang.reflect قرار دارند

عملگر instanceof

عملگر instanceof

- این عملگر مشخص می‌کند که آیا یک شیء، نمونه‌ای از یک نوع هست یا خیر
  - مقداری از نوع boolean برمی‌گرداند
- یک شیء (a) و یک کلاس و یا واسط (Type) می‌گیرد
  - اگر a نمونه‌ای از نوع Type (یا زیرکلاس آن) باشد، true برمی‌گرداند
  - توجه: رابطه **is a**
- معمولاً قبل از هر تغییر نوع به پایین (downcast) از این عملگر استفاده می‌کنیم
- مثال:

```
Person x = ...;
if(x instanceof Student)
  ((Student)x).register();
```

- در غیر این صورت، ممکن است خطای ClassCastException پرتاب شود

عملگر instanceof

```
Animal a ;
if(x>y) a = new Cat();
else  a = new Fish();
if(a instanceof Cat){
...
}else if(a instanceof Fish){
...
}
```

- مثال:

- نحوه استفاده از عملگر instanceof :

```
Ref a = ...
a instanceof Type
```

- قاعدتاً Type باید زیرکلاس Ref باشد
  - اگر Type همان کلاس Ref یا اَبَرکلاس Ref باشد: true برمی‌گرداند
  - اگر Type اَبَرکلاس، زیرکلاس یا خود Ref نباشد: خطای کامپایل (همیشه غلط)

```
String a = ...
a instanceof Integer
```

- نکته: اگر ارجاع موردنظر null باشد، این عملگر false برمی‌گرداند

مثال

```
String s = "Ali";
List<String> list;
list = new ArrayList<>();
boolean b;

b = s instanceof String;
b = s instanceof Serializable;
b = s instanceof List;
b = list instanceof Serializable;
b = list instanceof Collection;
b = list instanceof ArrayList;
b = list instanceof LinkedList;
b = list instanceof String;
b = list instanceof Collection<String>;
```

true

true

Syntax Error

true

true

true

false

Syntax Error

Syntax Error

بارگذاری پویا (Dynamic Loading)

بارگذاری پویا (Dynamic Loading)

- یک برنامه جاوا، از کلاس‌های مختلفی استفاده می‌کند
- اما همه این کلاس‌ها، در ابتدای اجرای برنامه در حافظه بارگذاری نمی‌شوند
- هر زمان که به یک کلاس نیاز شود، این کلاس در حافظه بارگذاری می‌شود
- در واقع در اولین استفاده از یک کلاس، آن کلاس بارگذاری می‌شود
- به این امکان در جاوا، بارگذاری پویا (**Dynamic loading**) می‌گویند

- سؤال: به ازای هر کلاس، دقیقاً چه چیزی در حافظه بارگذاری می‌شود؟

شیء کلاس (Class Object)

- به ازای هر کلاس، چه اطلاعاتی بارگذاری می‌شود؟
- هر کلاس، مشخصاتی دارد:
  - متدهای کلاس، پارامترها و مقادیر برگشتی متدها
  - ویژگی‌ها (فیلدها)، نوع هر ویژگی
  - مشخصات ویژگی‌ها و متدها (عمومی؟ استاتیک؟ ...)
- مجموعه مشخصات هر کلاس، در قالب یک شیء قابل نگهداری است
  - مثلاً یک شیء برای کلاس String، یک شیء برای کلاس Person ، یک شیء برای واسط List و ...
- شیئی که اطلاعاتی درباره یک کلاس دارد، «شیء کلاس» (**Class Object**) است
- در اولین استفاده از یک کلاس، یک «شیء کلاس» برای آن ساخته می‌شود
  - و در حافظه بارگذاری می‌شود

درباره بارگذاری پویا

- چه زمانی کلاس موردنظر بارگذاری می‌شود؟
  - در اولین استفاده، مثلاً:
  - هنگامی که اولین بار یک نمونه از آن ایجاد شود (با عملگر new)
  - و یا اولین بار که یک متد استاتیک از آن فراخوانی شود
- هنگام بارگذاری یک کلاس چه اتفاقاتی می‌افتد؟
  - یک شیء کلاس (Class Object) برای کلاس ایجاد و در حافظه بارگذاری می‌شود
  - فرایند مقداردهی اولیه متغیرهای استاتیک (static initialization)
- چه بخشی مسؤول بارگذاری کلاس جدید است؟
  - بخشی با نام بارگذار کلاس (**Class Loader**)
  - بارگذار کلاس مسؤول پیدا کردن کلاس موردنظر و بارگذاری آن در حافظه است

مثال

```
class Example {
    static int s1 = f();
    static {
        System.err.println("static block");
        s1 *= 2;
    }
    public static void g() {}
    private static int f() {
        System.err.println("inline static init");
        return 5;
    }
}
```

```
public class Statics {
    public static void main(String[] args) {
        Example e;
        System.err.println("After Declaration");
        Example.g();
        e = new Example();
        e = new Example();
    }
}
```

```
After Declaration
inline static init
static block
```

انواع بارگذار کلاس (Class Loader)

- **Bootstrap class loader**
- بخشی از JVM که به صورت سطح پایین (native) پیاده‌سازی شده است
- هسته اصلی جاوا را (از شاخه <JAVA_HOME>/jre/lib) بارگذاری می‌کند
- **Extensions class loader**
- کلاس‌های موجود در شاخه <JAVA_HOME>/jre/lib/ext را بارگذاری می‌کند
- **System class loader**
- کلاس‌های موجود در CLASS-PATH را می‌یابد و بارگذاری می‌کند
- **User-defined class loaders**
- برنامه‌نویس می‌تواند یک بارگذار (Class Loader) جدید معرفی کند
  (مثلاً برای دریافت اطلاعات کلاس از پایگاه داده یا از طریق شبکه)

شیء کلاس (Class Object)

شیء کلاس و متد getClass

- اولین بار که از یک کلاس استفاده می‌کنیم، این کلاس در حافظه بارگذاری می‌شود
- اطلاعات مربوط به این کلاس، در شیئی با نام «**شیء کلاس**» در حافظه جای می‌گیرد
- مثلاً یک شیء در حافظه اطلاعات کلاس String
  و شیء دیگری، اطلاعاتی درباره کلاس Person را نگهداری می‌کند
- هر شیء، یک ارجاع به «**شیء کلاس**» (Class Object) مربوط به کلاس خودش دارد
- این ارجاع با کمک متد **getClass()** برمی‌گردد
- متد getClass در Object پیاده‌سازی شده و final است

```
Animal a = new Dog("Fido");
String s = a.getClass().getSimpleName();
s = a.getClass().getName();
```

Metaspace و Permanent Generation

هر «شیء کلاس» در حافظه جای می‌گیرد. بخشی از حافظه مسؤول نگهداری این اشیاء است

قبل از نسخه ۸ جاوا

- اطلاعات کلاس‌ها (شیء کلاس‌ها) در بخشی به نام **PermGen** ذخیره می‌شود
- اگر پروژه بسیار بزرگی داشته باشیم، ممکن است این فضا پر شود و خطا ایجاد شود

OutOfMemoryError

- برنامه‌ای که کلاس‌های زیادی (کتابخانه‌ها و JAR های متنوع) را استفاده و بارگذاری کند

- حجم حافظه PermGen قابل تنظیم است:

java **-XX:MaxPermSize**=512m MyClass

بعد از جاوا ۸

اطلاعات مربوط به کلاس‌ها در **Metaspace** نگهداری می‌شود

- **PermGen حذف شده است**
- برخی از مشکلات و دردسرها هم از بین رفته: دیگر **نیازی به تنظیم PermSize نیست**

یادآوری: تنظیم اندازه حافظه Heap با کمک Xms– و Xmx–

راه‌های رسیدن به شیء کلاس

۱- استفاده از دستور **.class** بعد از نام کلاس

- مثال:

```
Class c = Person.class;
```

۲- استفاده از متد استاتیک **Class.forName**

```
Class c = Class.forName("ir.javacup.Person");
```

۳- فراخوانی متد **getClass** بر روی یک شیء

```
Object o = new Person();
Class c = o.getClass();
```

تفاوت ماهیت عملگر instanceof و شیء کلاس

- این دو دستور چه تفاوتی دارند؟

```
if(c instanceof Person)...
```

```
if(c.getClass().equals(Person.class))...
```

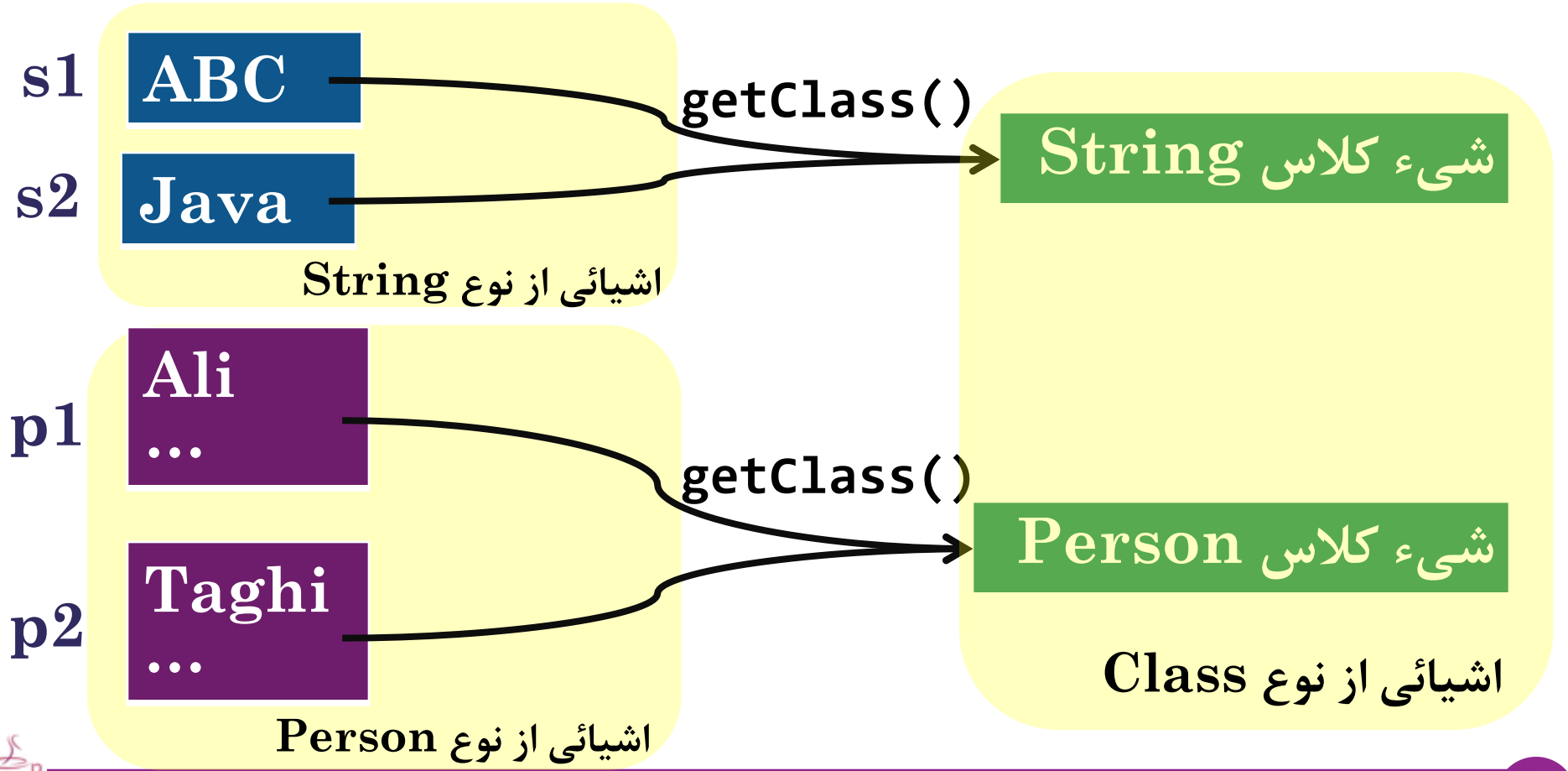
- دستور اول (عملگر instanceof) :
  - اگر c از نوع Person یا یکی از زیرکلاس‌های Person باشد، true برمی‌گرداند
  - رابطه is-a را بررسی می‌کند
- دستور دوم (استفاده از شیء کلاس) :
  - اگر c دقیقاً از نوع Person باشد، true برمی‌گرداند
- نکته: عملگر instanceof همانند متد isInstance() در Class است

```
if(Person.class.isInstance(c))...
```

امکانات کلاسِ Class

شیء کلاس

```
String s1 = "ABC";
String s2 = "Java";
Person p1 = new Person("Ali");
Person p2 = new Person("Taghi");
```



امکانات کلاسِ java.lang.Class

```
public final class Class<T> implements Serializable,...
```

- اطلاعاتی درباره متدهای کلاس موردنظر
  - فهرست متدها
  - دریافت یکی از متدها با کمک نام و مشخصات پارامترها
- فیلدهای کلاس موردنظر
  - فهرست فیلدها، دریافت یکی از فیلدها، ...
- سازنده‌ها (Constructor)
- اطلاعاتی درباره حاشیه‌نویسی‌ها (Annotation)
- ...

فرض

```
class Circle{
    public Double radius;
    public Circle() {}
    public Circle(Double radius) {
        this.radius = radius;
    }
    public Double getRadius() {
        return radius;
    }
    public void setRadius(Double radius) {
        this.radius = radius;
    }
    public double area(){
        return Math.PI*Math.pow(radius, 2);
    }
    public double perimeter(){
        return Math.PI*2*radius;
    }
}
```

کلاس Field

- با کمک شیئی از نوع Field اطلاعاتی درباره یک فیلد کسب می‌کنیم
  - مثلاً مقدار آن را دریافت کنیم یا تغییر دهیم

```
Circle circle = new Circle();
circle.radius = 2.5;
Class circleClass =
   Class.forName("ir.javacup.reflection.Circle");
Field[] fields = circleClass.getFields();
for (Field field : fields)
   if(field.getName().equals("radius")){
      Object value = field.get(circle);
      Double r = (Double) value;
      System.out.println(r);
      field.set(circle, r*2);
      System.out.println(circle.radius);
   }
```

مثال

```
Field[] fields = Circle.class.getFields();

Field field = fields[0];

field = Circle.class.getField("radius");

Circle circle = new Circle();

field.set(circle, 2.0);
```

کلاس Method

- با کمک شیئی از نوع Method می‌توانیم:

درباره یک متد صحبت کنیم، اطلاعاتی درباره آن کسب کنیم و آن را فراخوانی کنیم

```
Circle circle = new Circle();
Class circleClass = circle.getClass();
Method setter =
    circleClass.getMethod("setRadius", Double.class);
setter.invoke(circle, new Double(2.5));
Method getter = circleClass.getMethod("getRadius");
System.out.println(getter.invoke(circle));
```

مثال

```
Circle c = new Circle(1.5);
Method[] methods = c.getClass().getMethods();
Method method = methods[0];
method= c.getClass().getMethod("setRadius", Double.class);
method.invoke(c, 2.0);
Parameter[] params = method.getParameters();
for (Parameter param : params) {
   Class paramType = param.getType();
   System.out.println(paramType.getSimpleName());
}
method = c.getClass().getMethod("area");
Object result = method.invoke(c);
System.out.println((double)result);
```

کلاس Constructor

- امکان کار با یک سازنده (Constructor) را فراهم می‌کند
- متد newInstance از کلاس Class هم «سازنده بدون پارامتر» را فراخوانی می‌کند

```
Class circleClass = Circle.class;
Constructor cons =
    circleClass.getConstructor(Double.class);
Object o =  cons.newInstance(new Double(1.0));
Circle c = (Circle) o;
System.out.println(c.area());
Circle c2 = (Circle) circleClass.newInstance();
```

مثال

```
Class clazz =
    Class.forName("ir.javacup.reflection.Circle");
Constructor[] array = clazz.getConstructors();
Constructor cons = array[0];
cons = clazz.getConstructor(Double.class);
Object newObject = cons.newInstance(1.5);
cons = clazz.getConstructor();
newObject = cons.newInstance();
Circle inst = (Circle) newObject;
```

کلاس Annotation

```
@WebService
class Circle{
   @Deprecated
   @WebMethod
   public void f(){}
...
```

- امکان کار با حاشیه‌نویسی‌ها
  - حاشیه‌های یک کلاس، متد یا فیلد
- مثال:

```
Annotation[] annotations = Circle.class.getAnnotations();
for (Annotation annotation : annotations)
  System.out.println(annotation.annotationType());
annotations= Circle.class.getMethod("f").getAnnotations();
for (Annotation a : annotations)
  System.out.println(a.annotationType().getSimpleName());
```

```
interface javax.jws.WebService
Deprecated
WebMethod
```

کوییز

سؤال

- تفاوت اصطلاحات زیر چیست؟ (البته اصطلاحات انگلیسی مهمترند)
  - بارگذاری پویا (Dynamic Loading)
  - انقیاد پویا (Dynamic Binding)
- هر یک از موارد فوق به کدام‌یک از مباحث زیر مرتبط است؟
  - بازتاب (Reflection)
  - چندریختی (Polymorphism)

- نکته: پیوند پویا (Dynamic Linking) هم با موارد فوق متفاوت است

سؤال

- فرض کنید می خواهید RMI را پیاده‌سازی کنید
  - Remote Method Invocation
  - امکان فراخوانی متد از راه دور
  - فراخوانی متدی که در یک کامپیوتر دیگر اجرا خواهد شد
- از کدام‌یک از امکانات زیر استفاده خواهید کرد؟
  - Socket Programming
  - Serialization
  - Reflection

چند نکته

پرتاب خطا در اثر استفاده از بازتاب

- هنگام استفاده از بازتاب (Reflection) ممکن است خطاهایی پرتاب شود:
  - ClassNotFoundException : کلاس موردنظر وجود ندارد
  - NoSuchMethodException, : متد موردنظر وجود ندارد
  - IllegalAccessException : دسترسی مجاز نیست
    (مثلاً عضو موردنظر private است)
  - InvocationTargetException : فراخوانی متد موردنظر، خطا پرتاب کرده است
- بسیاری از خطاهایی که در حالت عادی کامپایلر می‌گرفت،
  در بازتاب تبدیل به Exception می‌شود
  - فرایند بازتاب، به طور کامل در زمان اجرا (runtime) انجام می‌شود

تفاوت getMethod و getDeclaredMethod

- متدهایی مثل getMethods, getMethod, getField, getFields

۱- اعضای به ارث‌رسیده را هم برمی‌گردانند

۲- فقط اعضای public کلاس را برمی‌گردانند

- متدهایی مثل getDeclaredMethods, getDeclaredMethod, getDeclaredField, getDeclaredFields

۱- فقط اعضایی که در همان کلاس تعریف شده‌اند را برمی‌گردانند
(اعضای به ارث‌رسیده را هم برنمی‌گردانند)

۲- اعضای غیرعمومی کلاس را هم برمی‌گردانند (private ، protected و ...)

اگر در استفاده از این اعضا، سطح دسترسی رعایت نشود: IllegalAccessException

بازتاب اعضای استاتیک

- برای استفاده از متدها و فیلدهای استاتیک، ذکر شیء لازم نیست

```
class X{
 public void f(){}
 public static void g(){}
 public int a;
 public static int b;
}
```

```
X x = new X();
X.class.getMethod("f").invoke(x);
X.class.getMethod("g").invoke(null);
X.class.getField("a").set(x, 1);
X.class.getField("b").set(null, 2);
```

بازتاب انواع داده اولیه (Primitive Data Types)

- فراخوانی class. روی انواع داده اولیه هم ممکن است
  - مثال: `int.class` ، `double.class` و `void.class`
  - کاربرد: توصیف پارامترها و مقدار برگشتی متدها

```
Method method =
  String.class.getMethod("substring", int.class);
```

```
Method m = Circle.class.getMethod("f");
if(m.getReturnType().equals(void.class))
 ...
```

کوییز

کوییز

- مقدار pCount و result در قطعه‌برنامه زیر چه خواهد بود؟

```
Class stringClass = String.class;
Method method ;
method = stringClass.getMethod("substring", int.class);
int pCount = method.getParameterCount(); 1
Object returned = method.invoke("Taghi Taghavi", 6);
String result = (String) returned; Taghavi
```

- معادل این برنامه بدون استفاده از بازتاب (Reflection) چگونه است؟

```
String result2 = "Taghi Taghavi".substring(6);
```

بازتاب و انواع عام

انواع عام در بازتاب

- کلاس Class و کلاس Constructor از انواع عام (Generic) هستند

```
Class<Circle> circleClass = Circle.class;
Constructor<Circle> cons =
    circleClass.getConstructor(Double.class);
Circle c1 = cons.newInstance(new Double(1.0));
Circle c2 = circleClass.newInstance();
```

```
Class clazz = Circle.class;
Object o2 = clazz.newInstance();
Circle c2 = (Circle) o2;
```

```
Class circleClass = Circle.class;
Constructor cons = circleClass.getConstructor();
Object o =  cons.newInstance(new Double(1.0));
Circle c = (Circle) o;
```

بازتاب انواع عام (Generic)

- فراخوانی class. روی پارامتر نوع در انواع عام (Generic) ممکن نیست
  - البته فراخوانی متد getClass() روی هر شیئی ممکن است
- مثال:

```
class GenericType<T>{
 private T element;
 public void f(){
   Class c2  = element.getClass();
   Class c1 = T.class;
 }
}
```

- چرا؟!
- می‌دانیم: پارامتر نوع (Type Parameter) در زمان اجرا حذف می‌شود
  - فرایند مَحو (Erasure)
- معلوم نیست شیئی از نوع List در زمان اجرا List<String> است یا List<Double>

```
void addInteger(ArrayList<String> list) throws Exception{
 Method m = list.getClass().getMethod("add", Object.class);
 m.invoke(list, new Integer(2));
}
```

```
void addObject(ArrayList<String> list) throws Exception{
 Method m = list.getClass().getMethod("add", Object.class);
 m.invoke(list, new Object());
}
```

```
ArrayList<String> list = new ArrayList<String>();
```

```
addObject(list);
for (Object o : list)
 System.out.println(o);
```

```
addInteger(list);
for (Object o : list)
 System.out.println(o);
```

```
addObject(list);
addInteger(list);
for (String s : list)
 System.out.println(s);
```

امکانات بازتاب

نکته

- در حالت عادی برای فراخوانی یک متد، تغییر یک فیلد، ایجاد شیء و ... از بازتاب (Reflection) استفاده نمی‌کنیم
- بازتاب فرایندی در زمان اجرا (runtime) است
  - خطای برنامه‌نویس را از زمان کامپایل به زمان اجرا منتقل می‌کند
  - سرعت اجرای یک متد به صورت مستقیم بیشتر از فراخوانی با کمک بازتاب است
- فقط زمانی که به بازتاب نیاز دارید از آن استفاده کنید (وقتی که مجبورید)
  - مثل نیاز به RMI که بدون بازتاب قابل پیاده‌سازی نیست
- البته امکاناتی هم در بازتاب هست که در حالت عادی وجود ندارد
  - مثلاً می‌توانیم یک متد private را فراخوانی کنیم!

تغییر سطح دسترسی

- با کمک بازتاب، امکان تغییر سطح دسترسی یک عضو وجود دارد
  - مثلاً یک متد private را public کنیم و آن را فراخوانی کنیم!
- محدوده اِعمال تغییر دسترسی، همان شیئی است که متد یا فیلد موردنظر را منعکس می‌کند

```
class MyClass{
  private void privateMethod(){...}
}
```

```
MyClass instance = new MyClass();
Method method =
 MyClass.class.getDeclaredMethod("privateMethod");
method.setAccessible(true);
method.invoke(instance);
```

مثال

```
public final class Integer extends Number
implements Comparable<Integer> {
 private final int value;
```

```
public static void swap(Integer first, Integer second) {
 try {
   Integer temp = new Integer(second);
   Field field = Integer.class.getDeclaredField("value");
   field.setAccessible(true);
   field.set(second, first.intValue());
   field.set(first, temp.intValue());
   field.setAccessible(false);
 } catch (Exception e) {e.printStackTrace();}
}
```

```
Integer a = new Integer(1);
Integer b = new Integer(7);
swap(a, b);
System.out.println(a);
System.out.println(b);
```

```
7
1
```

تمرین عملی

تمرین عملی

- تمرین ۱: گرفتن اطلاعات کلاس از کاربر
  - گرفتن نام متد
  - فراخوانی متد روی شیء موردنظر
- تمرین ۲:
  - ایجاد شیء
  - فراخوانی متد
  - تغییر مقدار متغیر
  - تغییر سطوح دسترسی

جمع‌بندی

جمع‌بندی

- مفهوم بازتاب (Reflection) در برنامه‌نویسی
  - مشاهده، استفاده و تغییر امکانات برنامه در زمان اجرا
  - امکاناتی مانند کلاس‌ها، متدها، فیلدها و ...
- امکانات جاوا در زمینه بازتاب
- مفهوم بارگذاری پویا (Dynamic Loading)
- مفهوم شیء کلاس (Class Object)

مطالعه کنید

- کتاب دایتل این مبحث را پوشش نداده است
- سایر مراجع:

- Thinking in Java (Fourth Edition), Bruce Eckel

  Type Information 393

- http://www.javatpoint.com/java-reflection
- http://tutorials.jenkov.com/java-reflection/index.html

تمرین

- برنامه‌ای بنویسید که از راه دور امکان فراخوانی متد را فراهم کند
  - به این کار RMI یا RPC می‌گویند
- Remote Method Invocation, Remote Procedure Call
  - البته امکانات این کار (RMI) در جاوا وجود دارد
  - برای سادگی:
    - در سمت سِرور یک لیست ایجاد کنید
    - در سمت کلاینت، اسم متد و پارامترهای آن را از کاربر بگیرید
    - دستور فراخوانی متد موردنظر را به سمت سرور ارسال کنید
    - خروجی متد موردنظر را از سمت سرور برای کلاینت بفرستید
  - برای این کار باید از موارد زیر استفاده کنید:
- Socket Programming, Serialization, Reflection

جستجو کنید و بخوانید

- کاربردهای بازتاب
  - کاربردها در Java Enterprise Edition
- امکانات جاوا برای بازتاب آرایه‌ها
  - ایجاد آرایه، دسترسی به عناصر و ...
  - کلاس java.lang.reflect.Array
- تعریف و ایجاد کلاس در زمان اجرا
  - برنامه‌نویسی در زمان اجرا !
- مفهوم پروکسی و کلاس java.lang.reflect.Proxy
  - برنامه‌نویسی جنبه‌گرا (Aspect Oriented Programming) با کمک بازتاب

پایان