JAVA PROGRAMMING BASICS

Module 3: Java Standard Edition

Training program

1. Java I/O Streams
2. Java Serialization
3. Java Database Connectivity
4. Java GUI Programming
5. The basics of Java class loaders
6. Reflections
7. Annotations
8. The proxy classes
9. Java Software Development
10. Garbage Collection

Module contents

- Reflection
  - The Java Reflection
  - The "Class" class
  - Retrieving Class Objects
  - Discovering Class Members
  - Dynamic invocation of methods
  - Using Java Reflection for access to private members

Module contents

- Reflection
  - The Java Reflection
  - The "Class" class
  - Retrieving Class Objects
  - Discovering Class Members
  - Dynamic invocation of methods
  - Using Java Reflection for access to private members

The Java Reflection 1/2

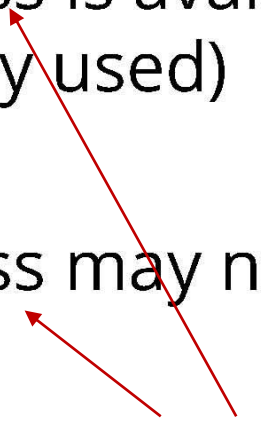
- Java provides two ways to discover information about an object at runtime

❑ **Traditional runtime class identification**

The object's class is available at compile and runtime (Most commonly used)

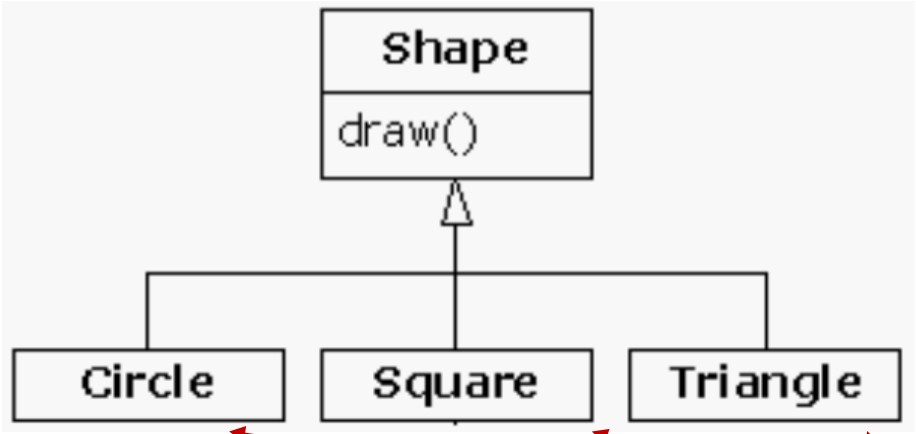
❑ **Reflection**

The object's class may not be available at compile or runtime



metaobject with metadata

Run-Time Type Identification example-1



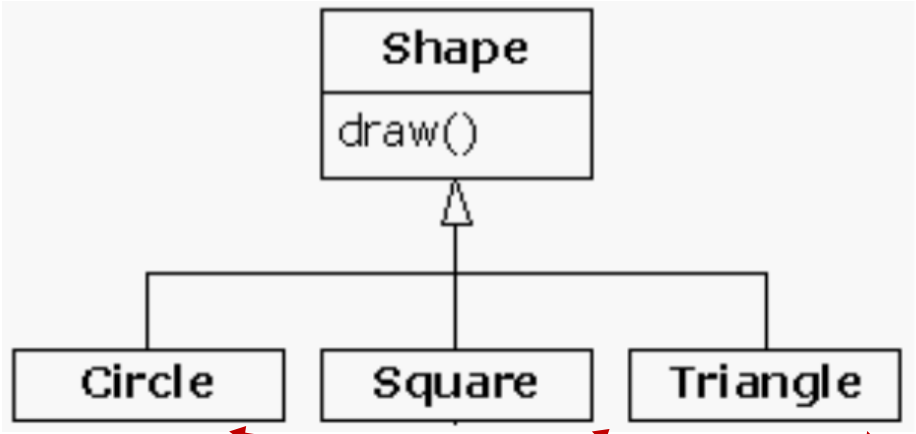
Overrides super.draw()

```
public static void main(String[] args) {
    Shape circle = new Circle();
    Shape square = new Square();
    Shape triangle = new Triangle();
    List<Shape> shapeList = Arrays.asList(circle, square, triangle);
    for (Shape shape : shapeList) {
        shape.draw();
    }
}
```

specific type of the created object is "forgotten"

casting to the required subtype by RTTI

Run-Time Type Identification example-2



Overrides super.draw()

```
public static void main(String[] args) {
    Shape circle = new Circle();
    Shape square = new Square();
    Shape triangle = new Triangle();
    System.out.println (triangle instanceof Shape);      //true
    System.out.println(triangle instanceof Triangle);    //true
```

instanceof tests if an object is of a given type

Don't use instanceof instead of polymorphism

The Java Reflection 2/2

- **Reflection** in a programming language context refers to the ability to observe and/or manipulate the inner workings of the environment programmatically

introspection

- The Reflection API in Java is used to view information about classes, interfaces, methods, fields, constructors, annotations during the execution of java programs. It is not necessary to know the names of the studied elements in advance.
- All classes for working with reflection are located in the **java.lang.reflect** package

The Java Reflection

Using the Java Reflection API, you can:

- Determine the class of the object.
- Get information about class modifiers, fields, methods, constructors, and superclasses.
- Find out which constants and methods belong to the interface.
- Create an instance of a class whose name is unknown until the time the program is executed.
- Get and set the value of a property of an object.
- Call a method on the object.
- Create a new array, the size and type of components of which are unknown until the execution of the programs.

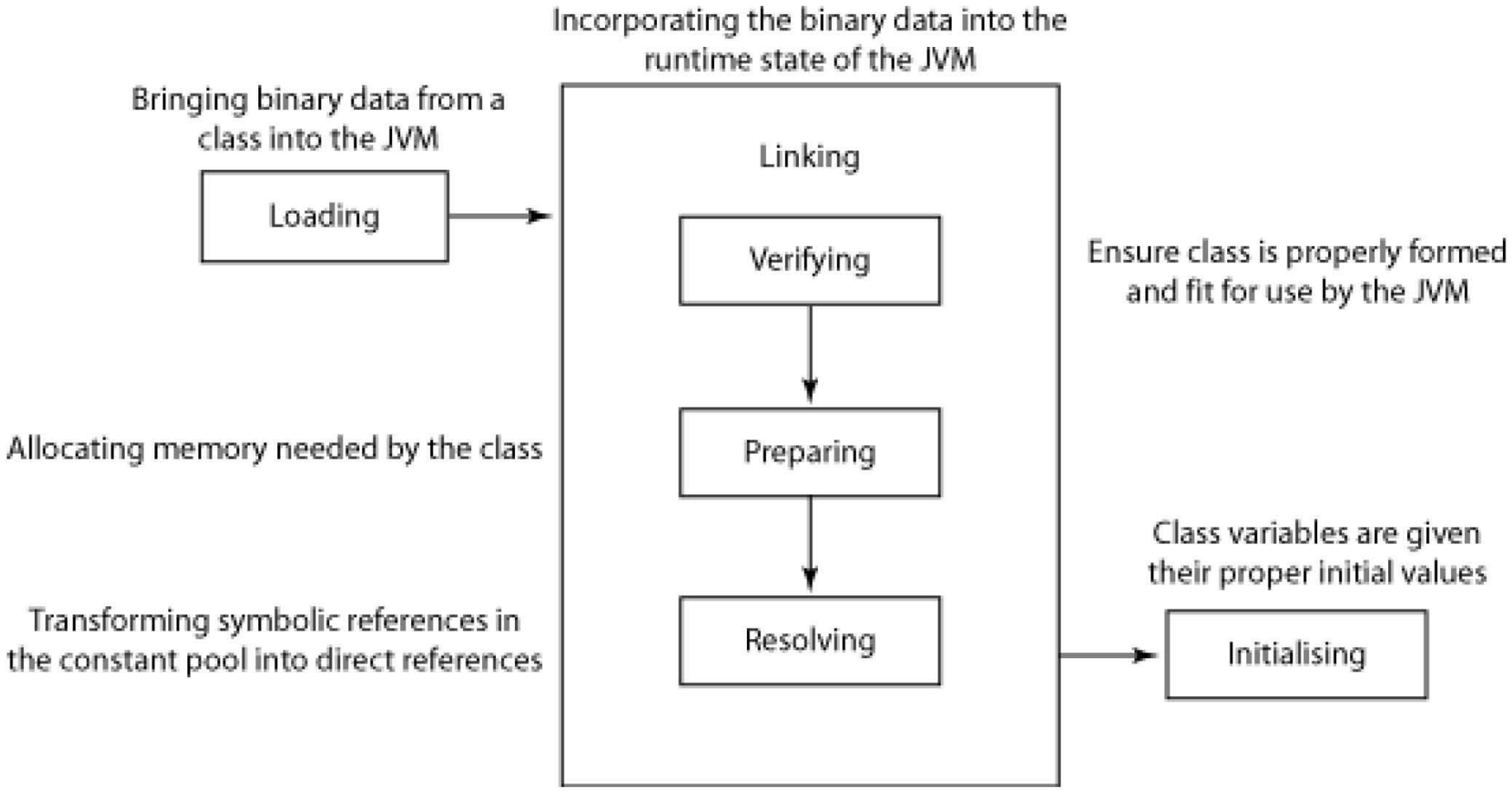
Module contents

- Reflection
  - The Java Reflection
  - The "Class" class
  - Retrieving Class Objects
  - Discovering Class Members
  - Dynamic invocation of methods
  - Using Java Reflection for access to private members

The "Class" class 1/5

- **Every class loaded into the JVM has a Class object corresponds to a .class file**
- The ClassLoader is responsible for finding and loading the class into the JVM
- **At object instantiation…**
- The JVM checks to see if the class is already loaded into the virtual machine
- Locates and loads the class if necessary
- Once loaded, the JVM uses the loaded class to instantiate an instance

The "Class" class 2/5



The "Class" class 3/5

- The JRE does not require that all classes are loaded prior to execution
- Class loading occurs when the class is first referenced
- All classes, interfaces, arrays, and primitive types have class literals
- Primitive types have corresponding wrapper classes

```
Class shapeClass = Shape.class;
Class booleanClass = boolean.class;
```

Wrappers Classes - 1

Литерал	Ссылка на объект Class
boolean.class	Boolean.TYPE
char.class	Character.TYPE
Byte.class	Byte.TYPE
short.class	Short.TYPE
int.class	Integer.TYPE
long.class	Long.TYPE
float.class	Float.TYPE
double.class	Double.TYPE
void.class	Void.TYPE

Wrappers Classes - 2

Void class using

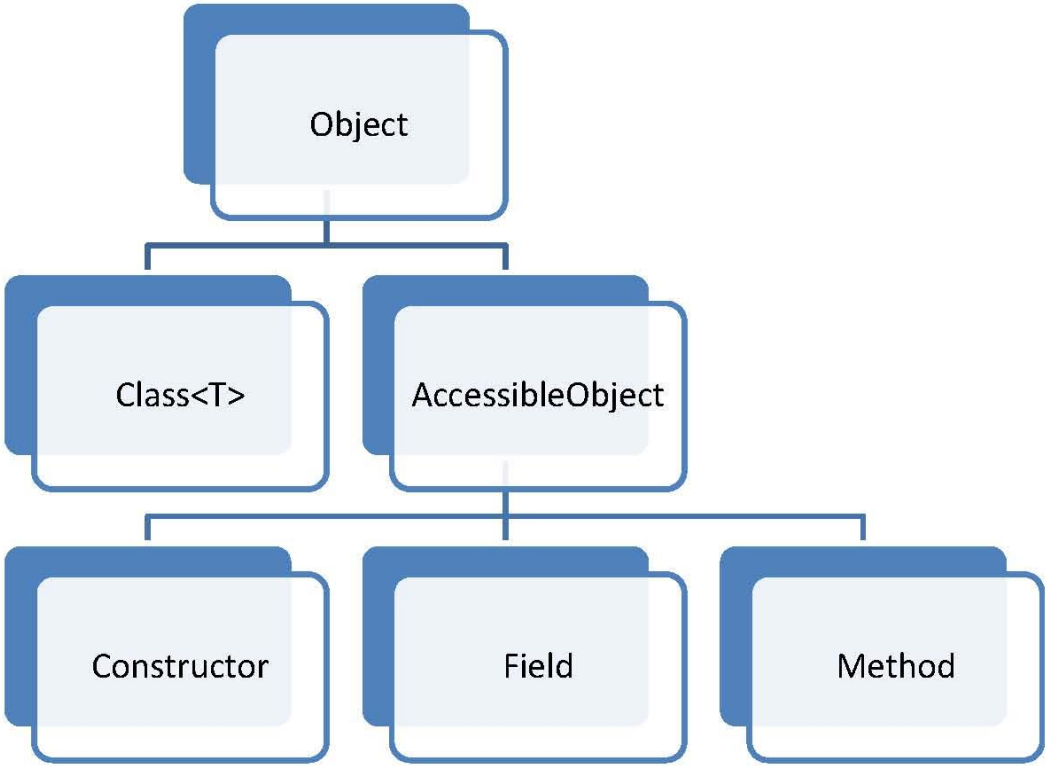
```
interface I<T> {
    T doStuff();
}

class A implements I<Void> {          //Use Void as generic
                                       //type parameter
    @Override
    public Void doStuff() {
        // ...
        return null;                  // anything else can not be returned
     }
}
```

```
Class c1 = A.class.getMethod(" doStuff ").getReturnType();
System.out.println(c1 == Void.TYPE);
System.out.println(c1 == Void.class);
```

The "Class" class 4/5

- The entry point for all reflection operations is java.lang.Class



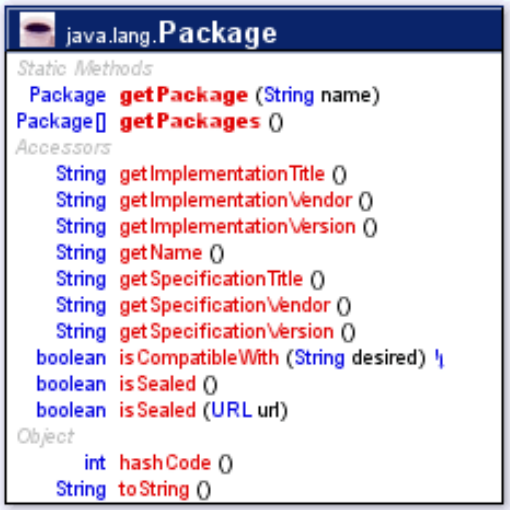
The "Class" class 5/5

- **java.lang.Class**
- Represents classes and interfaces within a running Java™ technology-based program
- **java.lang.reflect.AccessibleObject**
- The superclass for **Field**, **Method**, and **Constructor** classes
- **java.lang.Package**
- Provides information about a package that can be used to reflect upon a class or interface

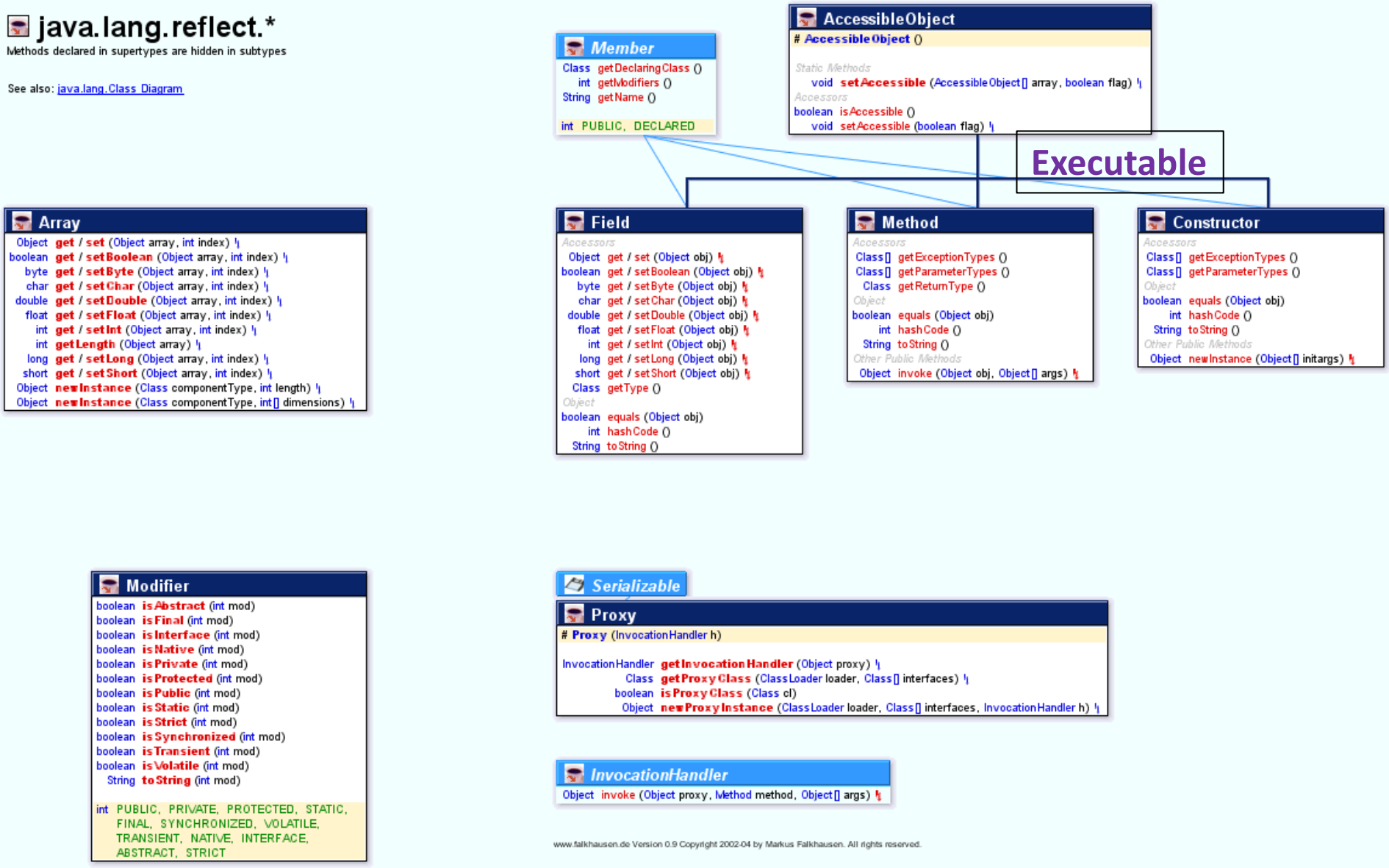
java.lang.Class

java.lang.* Class

See also: java.lang.reflect Diagram



java.lang.reflect



Module contents

- Reflection
  - The Java Reflection
  - The "Class" class
  - Retrieving Class Objects
  - Discovering Class Members
  - Dynamic invocation of methods
  - Using Java Reflection for access to private members

Retrieving Class Objects 1/4

- Get the Class object by invoking Object.getClass()

```
1. MyTestClass myTst = new MyTestClass();
2. final Class<?> cls = myTst.getClass();
3. int[] arr = new int[10];
4. Class<?> cls2 = arr.getClass();
```

see retrieveclass.RetrieveClassFromObjectDemo

Retrieving Class Objects 2/4

- Obtain a Class by appending ".class" to the name of the type (use if no object of the class)

1. **final** Class<?> cls = MyTestClass.**class**;

- For the primitive type being wrapped

1. **final** Class<?> cls = Double.***TYPE***;

```
Class<?> arrClass = int[][].class;
Class<?> booleanClass = boolean.class;
Class<?> voidType = Void.TYPE;
```

see retrieveclass.RetrieveClassFromClassDemo

Retrieving Class Objects 3/4

- Getting the corresponding Class using the static method Class.forName() (use if no object of the class and no class)

```
1. try {
2.     Class<?> cls =  Class.forName("classname");
3.     // ...
4. }
5. catch (ClassNotFoundException e) {
6.     //...
7. }
```

see retrieveclass.RetrieveClassByStringReference

Retrieving Class objects

```
/*double[] array Class instance*/
Class<?> doubleArrayClass = Class.forName("[D");
/*long[][] array Class instance*/
Class<?> longArrayClass = Class.forName("[[J");
/*String[][][] array Class instance*/
Class<?> stringArrayClass =
                                Class.forName("[[[Ljava.lang.String;");
```

Reference types:

L - prefix

; - suffix

in byte-code:

the dots . in the class FQN are replaced by /

the dots . in the inner classes are replaced by $

Primitive types:

B = byte	I = int
C = char	J = long
D = double	S = short
F = float	Z = boolean

Retrieving Class instance difference

```
public static void main(String[] args) {
    try {
        Shape shape = new Triangle();
        Class shapeClass = shape.getClass();   //RTTI (dynamic estimation)
        Class shapeClassAgain = Shape.class;    //static estimation
        Class shapeClassAgainToo = //static estimation
                                   Class.forName("typeinfoneed.Shape");
        System.out.println("shape.getClass() returns: "
                                   + shapeClass.getSimpleName());
        System.out.println("Shape.class returns: "
                                   + shapeClassAgain.getSimpleName());
       System.out.println("Class.forName(\"typeinfoneed.Shape\")
returns: " + shapeClassAgainToo.getSimpleName());
    } catch (ClassNotFoundException ex) {
    }
}
```

Retrieving Class instance difference

Output:
shape.getClass() returns: Triangle
Shape.class returns: Shape
Class.forName("typeinfoneed.Shape") returns: Shape

The getClass() method called from an object returns the runtime type, while .class and .forName () do a static estimation of the class !!!

Retrieving Class Objects 4/4

- Class.getSuperclass() Returns the super class for the given class

```
1. final Class<?> scls = Double.class.getSuperclass();
2. System.out.println(scls);
```

Console output:

class java.lang.Number

Inheritance introspection

java.lang.Class

Table 1.4 Methods of `Class` that deal with inheritance

Method	Description
Class[] **getInterfaces**()	Returns an array of `Class` objects that represent the direct superinterfaces of the target `Class` object
Class **getSuperclass**()	Returns the `Class` object representing the direct superclass of the target `Class` object or `null` if the target represents `Object`, an interface, a primitive type, or void
boolean **isAssignableFrom**(Class cls)	Returns `true` if and only if the class or interface represented by the target `Class` object is either the same as or a superclass of or a superinterface of the specified `Class` parameter
boolean **isInstance**(Object obj)	Returns `true` if and only if the specified `Object` is assignment-compatible with the object represented by the target `Class` object

see examineclass.ClassDeclarationSpy

Module contents

- Reflection
  - The Java Reflection
  - The "Class" class
  - Retrieving Class Objects
  - Discovering Class Members
  - Dynamic invocation of methods
  - Using Java Reflection for access to private members

Discovering Class Members 1/14

- Example class

```
1.  public class Main {
2.      static private final class MyTestClass {
3.          public String pubStrField = "TestStr";
4.          private int a = 7;
5.          protected long b = 8;
6.          public MyTestClass() {
7.          }
8.          public MyTestClass(int a) {
9.              this.a = a;
10.         }
```

- ...

Class that will be investigated

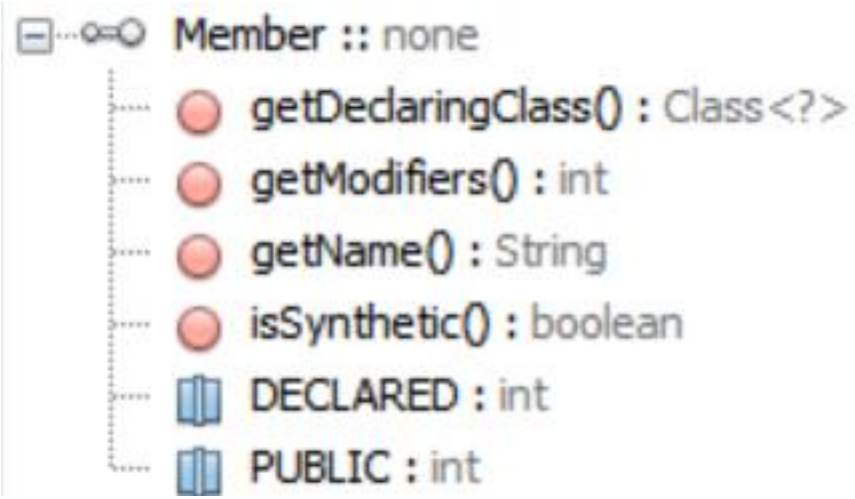
Discovering Class Members 2/14

- ...

```
1.  public MyTestClass(int a, long b) {
2.          this.a = a;
3.          this.b = b;
4.      }
5.      public int getA() {
6.          return a;
7.      }
8.      public long getB() {
9.          return b;
10.     }
11.     public void setA(int a) {
12.         this.a = a;
13.     }
14. }
}
```

Discovering Class Members 3/14

- **java.lang.reflect.Member**
- Interface that reflects identifying information about a single member (a field or a method) or a constructor



java.lang.reflect.Member interface

Table 2.3 Methods declared by the interface `Member`

Method	Description
Class **getDeclaringClass**()	Returns the `Class` object that declared the member
String **getName**()	Returns the name of the member
int **getModifiers**()	Returns the modifiers for the member encoded as an `int`

Discovering Class Members 4/14

- **java.lang.reflect.Modifier**
- The Modifier class provides static methods and constants to decode class and member access modifiers. The sets of modifiers are represented as integers with distinct bit positions representing different modifiers.

- **int** mods = cls.getModifiers();

see examineclass.ClassMembersSpy

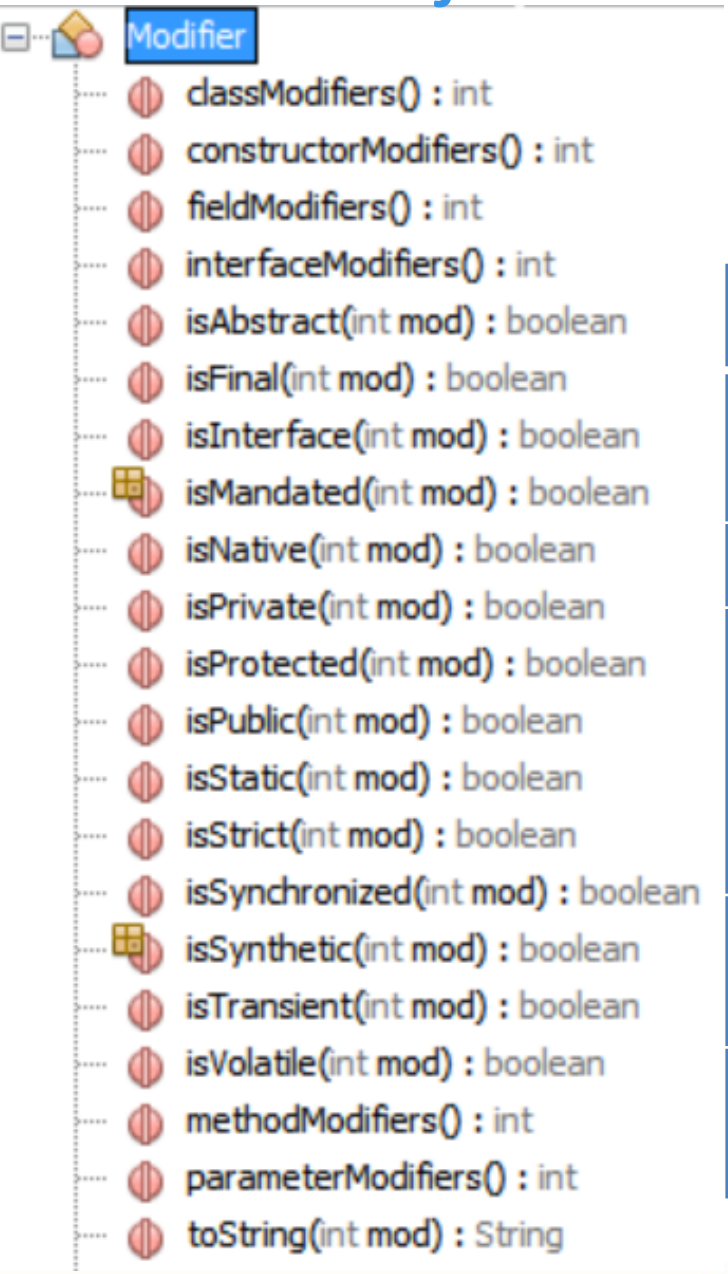
Discovering Class Members 6/14

```
1.  System.out.println("Modifiers: ");
2.  int mods = cls.getModifiers();
3.  if (Modifier.isPrivate(mods))
4.      System.out.println("private ");
5.  if (Modifier.isStatic(mods))
6.      System.out.println("static ");
7.  if (Modifier.isAbstract(mods))
8.      System.out.println("abstract ");
9.  if (Modifier.isFinal(mods))
10.     System.out.println("final ");
```

Console output:

```
Modifiers:
private
static
final
```

java.lang.reflect.Modifier class



Class modifiers

Flag Name	Value	Interpretation
ACC_PUBLIC	0x0001	Declares a public access type
ACC_FINAL	0x0010	Forbids inheriting
ACC_SUPER	0x0020	When an instruction is invoked, invokespecial treats the methods of the superclass in a special way.
ACC_INTERFACE	0x0200	Is an interface, not a class
ACC_ABSTRACT	0x0400	Doesn't allow you to instantiate a class

Discovering Class Members 5/14

Constant mane	Method	Modifier
ABSTRACT	isAbstract	abstract
FINAL	isFinal	final
INTERFACE	isInterface	interface
NATIVE	isNative	native
PRIVATE	isPrivate	private
PROTECTED	isProtected	protected
PUBLIC	isPublic	public
STATIC	isStatic	static
STRICT	isStrict	strictfp
SYNCHRONIZED	isSynchronized	synchronized
TRANSIENT	isTransient	transient
VOLATILE	isVolatile	volatile

java.lang.reflect.Modifier Field modifiers

Flag Name	Value	Interpretation
ACC_PUBLIC	0x0001	Declared public; may be accessed from outside its package.
ACC_PRIVATE	0x0002	Declared private; usable only within the defining class.
ACC_PROTECTED	0x0004	Declared protected; may be accessed within subclasses.
ACC_STATIC	0x0008	Declared static.
ACC_FINAL	0x0010	Declared final; never directly assigned to after object construction (JLS §17.5).
ACC_VOLATILE	0x0040	Declared volatile; cannot be cached.
ACC_TRANSIENT	0x0080	Declared transient; not written or read by a persistent object manager.
ACC_SYNTHETIC	0x1000	Declared synthetic; not present in the source code.
ACC_ENUM	0x4000	Declared as an element of an enum.

java.lang.reflect.Modifier Method modifiers

Flag Name	Value	Interpretation
ACC_PUBLIC	0x0001	may be accessed from outside its package.
ACC_PRIVATE	0x0002	accessible only within the defining class.
ACC_PROTECTED	0x0004	may be accessed within subclasses.
ACC_STATIC	0x0008	Declared static.
ACC_FINAL	0x0010	must not be overridden
ACC_SYNCHRONIZED	0x0020	invocation is wrapped by a monitor use.
ACC_BRIDGE	0x0040	A bridge method, generated by the compiler.
ACC_VARARGS	0x0080	Declared with variable number of arguments.
ACC_NATIVE	0x0100	implemented in a language other than Java.

java.lang.reflect.Modifier Method modifiers

Flag Name	Value	Interpretation
ACC_ABSTRACT	0x0400	no implementation is provided.
ACC_STRICT	0x0800	floating-point mode is FP-strict.
ACC_SYNTHETIC	0x1000	not present in the source code.

Discovering Class Members 7/14

- **java.lang.reflect.Field**
- A Field provides information about, and dynamic access to, a single field of a class or an interface
- The reflected field may be a class (static) field or an instance field.

```
Field f = cls.getField(fieldName);
Class<?> clazz = f.getType();
int foundMods = f.getModifiers();
```

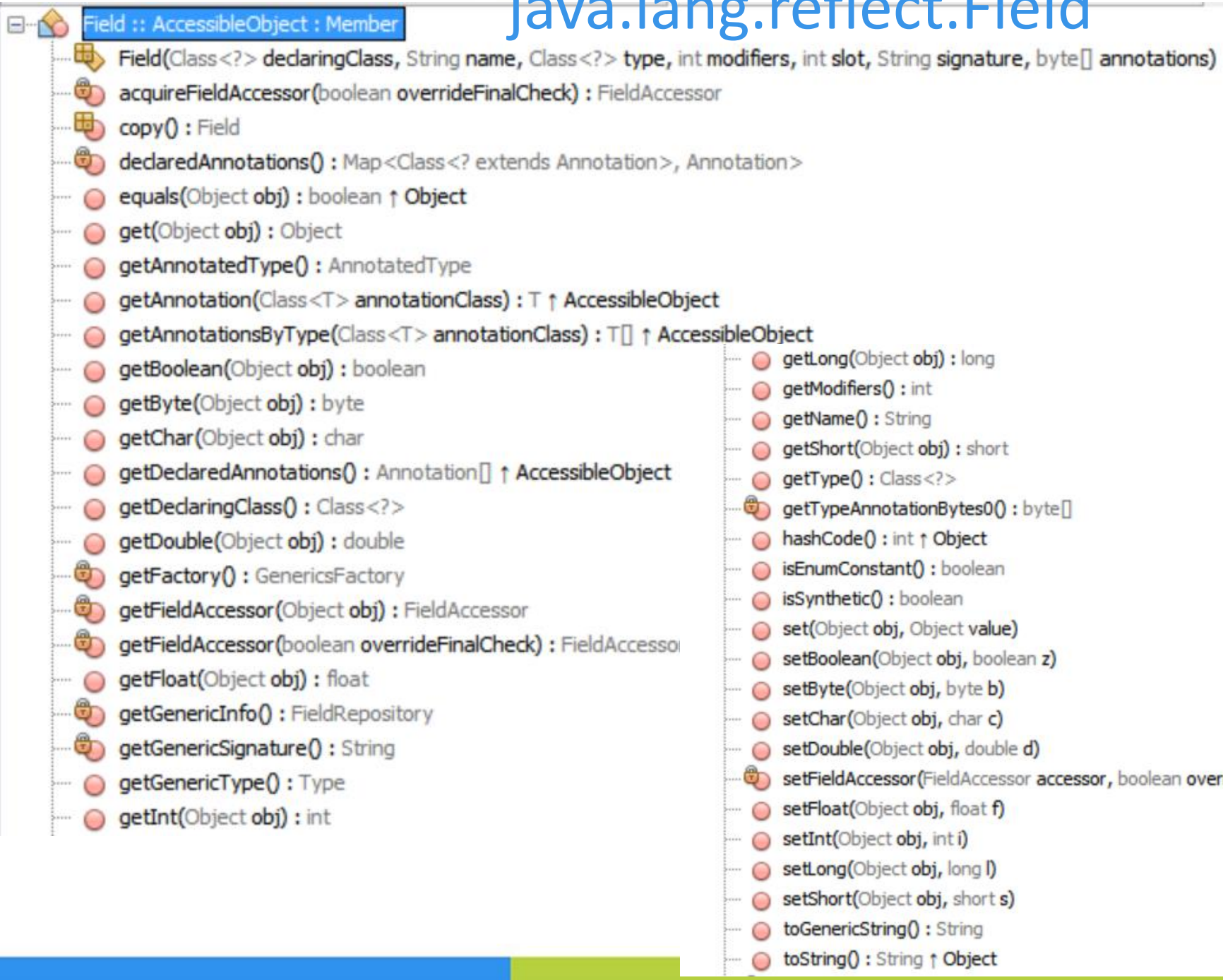
java.lang.reflect.Field

java.lang.Class

Table 2.1 Methods of `Class` for field introspection

Method	Description
Field **getField**(String name)	Returns a `Field` object that represents the specified public member field of the class or interface represented by this `Class` object
Field[] **getFields**()	Returns an array of `Field` objects that represents all the accessible public fields of the class or interface represented by this `Class` object
Field **getDeclaredField**(String name)	Returns a `Field` object that represents the specified declared field of the class or interface represented by this `Class` object
Field[] **getDeclaredFields**()	Returns an array of `Field` objects that represents each field declared by the class or interface represented by this `Class` object

java.lang.reflect.Field



java.lang.reflect.Field

Table 2.2 Methods defined by `Field`

Method	Description
Class **getType**()	Returns the `Class` object that represents the declared type for the field represented by this `Field` object
Class **getDeclaringClass**()	Returns the `Class` object that declared the field represented by this `Field` object
String **getName**()	Returns the name of the field represented by this `Field` object
int **getModifiers**()	Returns the modifiers for the field represented by this `Field` object encoded as an `int`
Object **get**(Object obj)	Returns the value in the specified object of the field represented by this `Field`
boolean **getBoolean**(Object obj) ...	Returns the value in the specified object of the boolean field represented by this `Field`

continued on next page

java.lang.reflect.Field

Table 2.2 Methods defined by `Field` *(continued)*

Method	Description
void **set**(Object obj, Object value)	Sets the field of the specified object represented by this `Field` object to the specified new value
void **setBoolean**(Object obj, boolean value) ...	Sets the field of the specified object represented by this `Field` object to the specified boolean value

Discovering Class Members 8/14

```
1. System.out.println("Public fields:");
2. Field[] fields = cls.getFields(); //returns only public fields
3. for (Field field : fields) {
4.     Class<?> fType = field.getType();\
5.     System.out.println("\tName: " + field.getName());
6.     System.out.println("\tType: " + fType.getName());
7. }
```

Console output:

```
Public fields:
        Name: pubStrField
        Type: java.lang.String
```

Discovering Class Members 9/14

```
1. System.out.println("All fields:");
2. fields = cls.getDeclaredFields();   //returns all fields
3. for (Field field : fields) {
4.     Class<?> fType = field.getType();
5.     System.out.println("\tName: " + field.getName());
6.     System.out.println("\tType: " + fType.getName());
7. }
```

see fieldexamine.FieldSpy<T>
and fieldexamine.setValue.Book

Discovering Class Members 10/14

- **java.lang.reflect.Constructor**
- Provides information about, and access to, a single constructor for a class
- Constructor permits widening conversions to occur when matching the actual parameters to newInstance() with the underlying constructor's formal parameters, but throws an IllegalArgumentException if a narrowing conversion would occur.

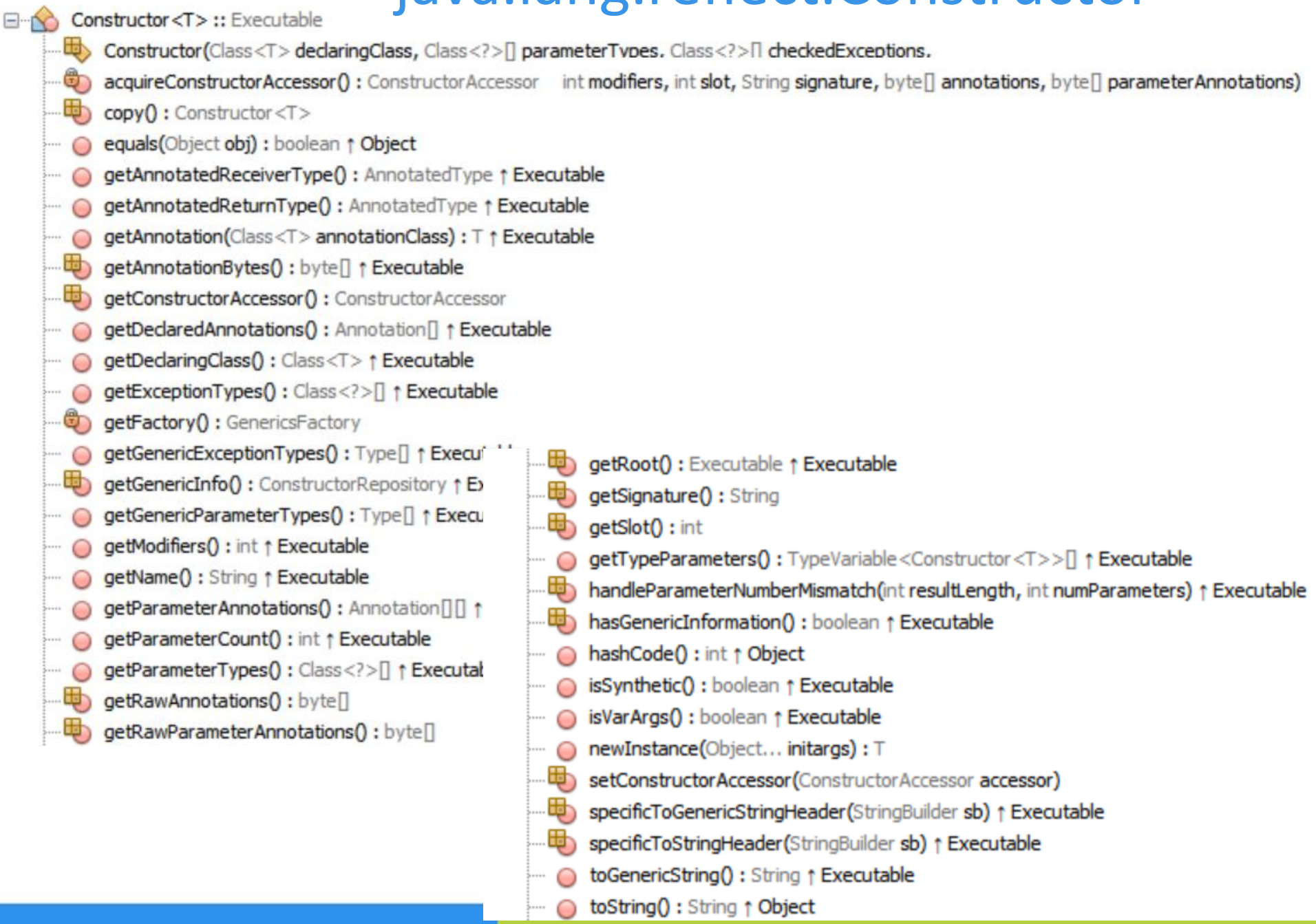
java.lang.reflect.Constructor

java.lang.Class

Table 3.2 Methods of Class for constructor introspection

Method	Description
Constructor **getConstructor**(Class[ ] parameterTypes)	Returns the public constructor with specified argument types if one is supported by the target class
Constructor **getDeclaredConstructor**(Class[ ] parameterTypes)	Returns the constructor with specified argument types if one is supported by the target class
Constructor[] **getConstructors**()	Returns an array containing all of the public constructors supported by the target class
Constructor[] **getDeclaredConstructors**()	Returns an array containing all of the constructors supported by the target class

java.lang.reflect.Constructor



java.lang.reflect.Constructor

Table 3.3 Reflective methods of `Constructor`

Method	Constructor
Class **getDeclaringClass**()	Returns the class object that declares the constructor represented by this `Constructor`
Class[ ] **getExceptionTypes**()	Returns a `Class` array representing the types of exceptions that can be thrown from the body of this `Constructor`
int **getModifiers**()	Returns a bit vector encoding the modifiers present and absent for this member
String **getName**()	Returns the name of the constructor

continued on next page

java.lang.reflect.Constructor

Table 3.3 Reflective methods of `Constructor` *(continued)*

Method	Constructor
Class[ ] **getParameterTypes**()	Returns a `Class` array representing the parameter types that are accepted by this constructor in order
Object **newInstance**(Object[ ] initargs)	Invokes the constructor with the specified parameters and returns the newly constructed instance

Discovering Class Members 11/14

```
1.  Constructor<?>[] constrs = cls.getConstructors();
2.  int i = 0;                          //returns only public constructors
3.  for (Constructor<?> ctr : constrs) {
4.      System.out.print("Constructor " + (i++) + " : ");
5.      Class<?>[] paramTypes = ctr.getParameterTypes();
6.      for (Class<?> paramType : paramTypes) {
7.          System.out.print(paramType.getName() + " ");
8.      }
9.      System.out.println();
10. }
```

Console output:

```
Constructor 0 :
Constructor 1 : int long
Constructor 2 : int
```

getDeclaredConstructors() returns all constructors

Discovering Class Members 12/14

```
1. try {
2.     Class<?>[] paramTypes = new Class<?>[] { int.class };
3.     Constructor<?> ctr = cls.getConstructor(paramTypes);
4.     MyTestClass t = (MyTestClass) ctr.newInstance(1);
5.     System.out.println("Fields: " + t.getA() + ", " + t.getB());
6. } catch (Exception ex) {
7.     ex.printStackTrace();
8. }
```

Console output:

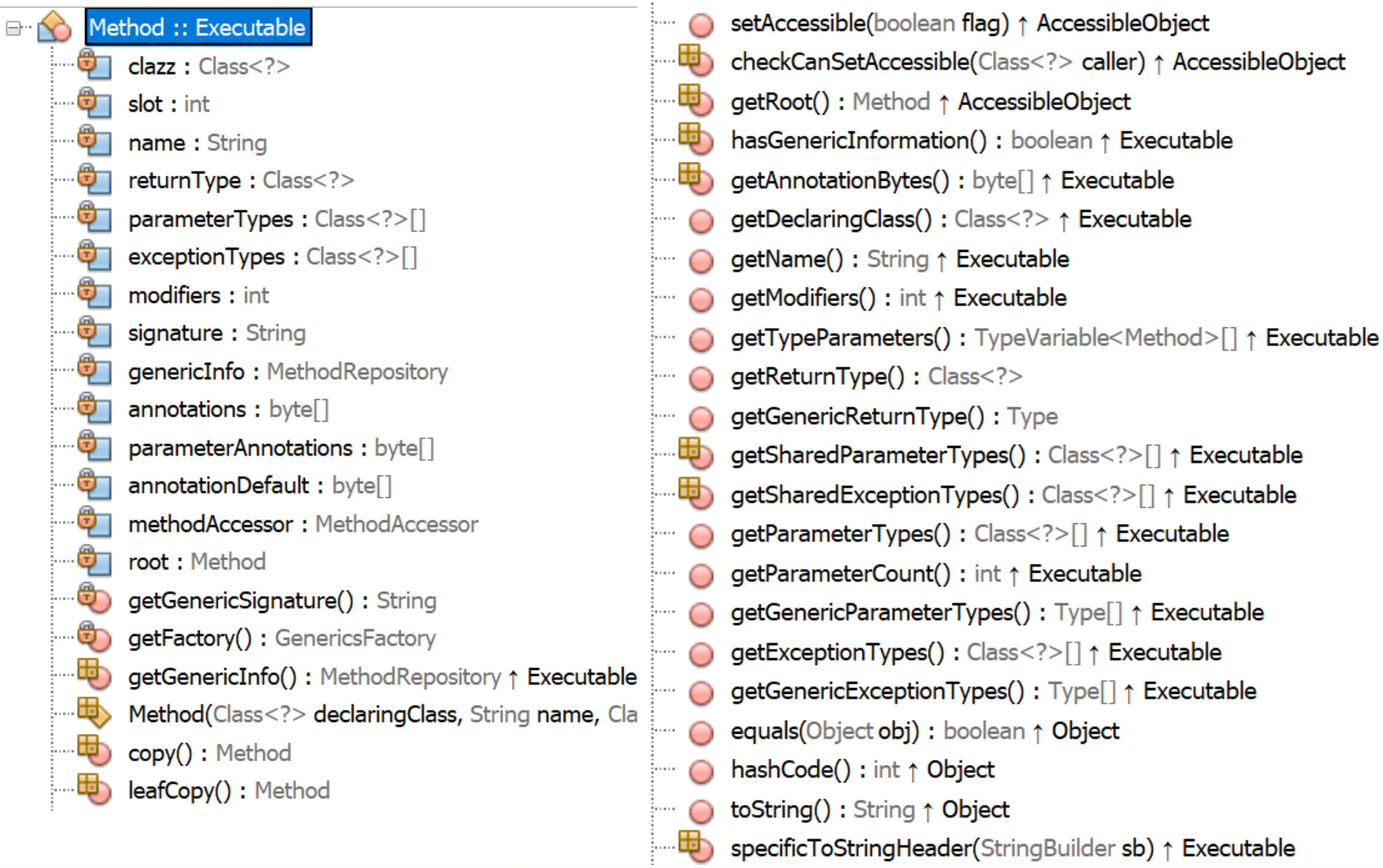
Fields: 1, 8

see constructor.ConstructorModifier
and constructor.FindConstructorByParameterType

Discovering Class Members 13/14

- **java.lang.reflect.Method**
- Provides information about, and access to, a single method on a class or interface

java.lang.reflect.Method



java.lang.reflect.Method

- invoke(Object obj, Object... args) : Object
- isBridge() : boolean
- isVarArgs() : boolean ↑ Executable
- isSynthetic() : boolean ↑ Executable
- isDefault() : boolean
- acquireMethodAccessor() : MethodAccessor
- getMethodAccessor() : MethodAccessor
- setMethodAccessor(MethodAccessor accessor)
- getDefaultValue() : Object
- getAnnotation(Class<T> annotationClass) : T ↑ Executable
- getDeclaredAnnotations() : Annotation[] ↑ Executable
- getParameterAnnotations() : Annotation[][] ↑ Executable
- getAnnotatedReturnType() : AnnotatedType ↑ Executable
- handleParameterNumberMismatch(int resultLength, int numParameters) : boolean ↑ Executable

java.lang.reflect.Method

java.lang.Class

Table 1.1 The methods defined by `Class` for method query

Method	Description
Method **getMethod**(String name, Class[] parameterTypes)	Returns a `Method` object that represents a public method (either declared or inherited) of the target `Class` object with the signature specified by the second parameters
Method[] **getMethods**()	Returns an array of `Method` objects that represent all of the public methods (either declared or inherited) supported by the target `Class` object
Method **getDeclaredMethod**(String name, Class[] parameterTypes)	Returns a `Method` object that represents a declared method of the target `Class` object with the signature specified by the second parameters
Method[] **getDeclaredMethods**()	Returns an array of `Method` objects that represent all of the methods declared by the target `Class` object

Discovering Class Members 14/14

```
1.  Method[] methods = cls.getMethods();
2.  for (Method method : methods) {
3.      System.out.println("Name: " + method.getName());
4.      System.out.println("\tReturn type: "
                + method.getReturnType().getName());
5.      Class<?>[] paramTypes = method.getParameterTypes();
6.      System.out.print("\tParam types:");
7.      for (Class<?> paramType : paramTypes) {
8.          System.out.print(" " + paramType.getName());
9.      }
10.     System.out.println();
11. }
```

java.lang.reflect.Method

java.lang.Class

Table 1.3 Methods defined by `Method`

Method	Description
Class **getDeclaringClass**()	Returns the `Class` object that declared the method represented by this `Method` object
Class[] **getExceptionTypes**()	Returns an array of `Class` objects representing the types of the exceptions declared to be thrown by the method represented by this `Method` object
int **getModifiers**()	Returns the modifiers for the method represented by this `Method` object encoded as an `int`
String **getName**()	Returns the name of the method represented by this `Method` object
Class[] **getParameterTypes**()	Returns an array of `Class` objects representing the formal parameters in the order in which they were declared
Class **getReturnType**()	Returns the `Class` object representing the type returned by the method represented by this `Method` object
Object **invoke**(Object obj, Object[] args)	Invokes the method represented by this `Method` object on the specified object with the arguments specified in the `Object` array

see methodexamine.MethodSpy

Module contents

- Reflection
  - The Java Reflection
  - The "Class" class
  - Retrieving Class Objects
  - Discovering Class Members
  - Dynamic invocation of methods
  - Using Java Reflection for access to private members

Dynamic invocation of methods 1/2

```
1.  try {
2.      MyTestClass obj = new MyTestClass();
3.      Class<?>[] paramTypes = new Class<?>[] { int.class };
4.      Method method = cls.getMethod("setA", paramTypes);
5.      Object[] arguments = new Object[] { Integer.valueOf(5) };
6.      method.invoke(obj, arguments);
7.      System.out.println("A: " + obj.getA());
8.  } catch (Exception ex) {
9.      ex.printStackTrace();
10. }
```

primitive type parameter

arguments = null, if the method has no arguments

Returns an object that must be cast to the type of the result returned by the method (for primitive types - a wrapper class, for void - null is returned).

Dynamic invocation of methods 2/2

Parameters-arrays representation → Object[].class
int[].class
double[][].class

```
1. try {
2.     MyTestClass obj = new MyTestClass();
3.     Class<?>[] paramTypes = new Class<?>[] { int.class };
4.     Method method = cls.getMethod("noneMethod",
   paramTypes);
5.     Object[] arguments = new Object[] { Integer.valueOf(5) };
6.     method.invoke(obj, arguments);
7. } catch (Exception ex) {
8.     System.out.println(ex.toString());
9. }
```

java.lang.NoSuchMethodException

see - methodexamine.invoke.MyTestMethodInvoke
and methodexamine.invoke.MethodInvoke<T>

Module contents

- Reflection
  - The Java Reflection
  - The "Class" class
  - Retrieving Class Objects
  - Discovering Class Members
  - Dynamic invocation of methods
  - Using Java Reflection for access to private members

Using Java Reflection for access to private members

disables all runtime access checks on uses of the metaobject referred to by **`field`**

```
1.  try {
2.      MyTestClass obj = new MyTestClass();
3.      Field field = cls.getDeclaredField("a");
4.      field.setAccessible(true);
5.      System.out.println("Private field value: " +
    field.getInt(obj));
6.      field.setInt(obj, 100);
7.      System.out.println("New private field value: " +
    field.getInt(obj));
8.  } catch (Exception ex) {
9.      ex.printStackTrace();
10. }
```

Console output

Private field value: 7

New private field value: 100

see fieldexamine.PrivateFieldAccess