The “Class” Class

The `Class` Class

- Browsing the Java 2 SE API you will find the `Class` class. A class that its name is “Class”.
  http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Class.html
- Each one of the values that exist during our code execution is either a primitive type value (e.g. int, double, float, byte etc.) or a class type reference (any reference for any object is a class type reference).
- For every class type reference, the JVM maintains an immutable instance of the class `java.lang.Class`.

The `Class` Class

- All objects extend (either directly or indirectly) from the `Object` class. Therefore, any of the methods that were declared within the `Object` class can be called on every object of every type.
- One of the methods we can find in `Object` is `getClass()`.

  http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Object.html
- For any object on which we call the `getClass()` method we will get a reference for an object of type `Class` that describes the class type of our object (the object on which we called the `getClass()` method).

The `Class` Class

- Getting a reference for a `Class` object is the first step for using any of the reflection capabilities.
- Once we get the reference for a `Class` object we can call any of the methods that were declared in `Class` on that object.
- Various methods that were declared in `Class` allow us getting detailed information about all members (methods, variables and constructors) of that class. Including the private ones.
- The methods and classes that belong to the Reflection API even provide a way to indirectly call methods on the object we currently inspect.

Retrieving a `Class` Object

- Getting a reference for a `Class` object is the first step for using any of the reflection capabilities.
- Once we get the reference for a `Class` object we can call any of the methods that were declared in `Class` on that object.
- Various methods that were declared in `Class` allow us getting detailed information about all members (methods, variables and constructors) of that class. Including the private ones.
- There are several ways for retrieving a `Class` object that describes a class we are interested at. The following slides describe these ways.

The `Object.getClass()` Method

- The `getClass()` method, that was declared in `Object`, can be invoked on every object.
- When calling this method on a specific object we get a reference for a Class object that describes the class from which that specific object was instantiated.

```
String str = new String("abc");
Class strClass = str.getClass();
```

- Calling the `getClass()` method we can get a `Class` object that describes a class from which an object was instantiated only.

The `Object.getClass()` Method

- We can't use this technique to get a `Class` object that describes an interface.
- We can use this technique to retrieve a `Class` object that describes a class that so far it was less clear for us that our object was instantiated from.

```
int []vec = new int[10];
Class vecClass = vec.getClass();
```

The `.class` Syntax

- Appending `.class` to the name of a class type returns a reference to the Class object that describes the class type we used its name.

```
Class stringClass = String.class;
```

- You can use this special syntax to get even the Class object that describes an interface class type.

```
Class cloneableClass = Cloneable.class;
```

The `Class.forName()` Method

- The `Class` class includes the `forName()` static method. Calling this method sending a full qualified name of a class type returns a reference to an object of type Class that describes the class type.

```
Class stringClass = Class.forName("java.lang.String");
```

Primitive Types

- Each primitive type (e.g. double, float etc..) has a class that represents it at runtime.
- These classes, that represent the primitive types, are separated from the wrapper classes (e.g. `Double`, `Float` etc..).
- While the wrapper classes are used to bridge between the primitive types world and the object world, the primitive types' classes are used to represent the primitive type values' types within the running code.

Primitive Types

- One of the possible ways for getting a `Class` object that represents a primitive type's class is by using the ".class" syntax. Writing `.class` preceding with the primitive type name will provide us with a reference to a `Class` object that represents the primitive type's class.

```
Class doubleClass = double.class;
```

- An alternative way can be using the `TYPE` static field we can find in each one of the wrapper classes.

```
Class doubleClass = Double.TYPE;
```

Class Methods

- The `Class` class includes several methods that when we call them we get a reference to object of type Class.

  `Class.getSuperclass()`

  Calling this method returns a reference to an object of type `Class` that describes the super class.

  `Class.getClasses()`

  Calling this methods returns an array of references to objects of type `Class` representing all the public classes and interfaces that are members of the class represented by this `Class` object.

Class Methods

`Class.getDeclaredClasses()`

Calling this methods returns an array of references to objects of type `Class` representing all classes and interfaces that are members of the class represented by this `Class` object.

`Class.getEnclosingClass()`

Calling this methods returns a reference to object of type `Class` representing the immediately enclosing class.

`Method`,`Constructor` & `Field`

- The `Method`, `Constructor` & `Field` classes represent class members.
- All of these classes include the following method:

  `Class.getDeclaringClass()`

  Calling this method returns a reference to an object of type `Class` that describes the class this member belongs to.

Class Modifiers & Types

- A class can be declared with modifiers. The possible modifiers include the following:

  Access Modifiers – public, private & protected

  Abstract Modifier – abstract

  Static Modifier – static (you can declare an inner class as abstract)

  Final Modifier – final

  Strict Floating Point Behavior – strictfp

- The `Class` class includes the method `getModifiers()`. Calling that method on a Class object returns an integer number that each one of its bits (turned on/off) tells something about the class modifiers

Class Modifiers & Types

- The class Modifier includes methods that can assist decoding `getModifiers()` returned value. This class also includes final fields for each one of the possible modifiers values.

  http://java.sun.com/j2se/1.5.0/docs/api/java/lang/reflect/Modifier.html

```
Class stringClass = String.class;
System.out.println(Modifier.toString(stringClass.getModifiers()));
```

Discovering Class Members

- Once we retrieve a `Class` object that describes the class we want to analyze, all that is left is calling various methods on that object in order to get objects that describe the class members.
- The class members are described by objects instantiated from the following classes: `Method`, `Field` & `Constructor`.
- Once we retrieve an object that describes one of our class members, we can call various methods on that specific object in order to get more information on that member.

Discovering Class Members

- Methods That Return List of Public Members Only

  (Excluding The Inherited Ones)

  Field[] getFields()

  Method[] getMethods()

  Constructor[] getConstructors()

- Methods That Return List of All Members

  (Excluding The Inherited Ones)

  Field[] getDeclaredFields()

  Method[] getDeclaredMethods()

  Constructor[] getDeclaredConstructors()

Discovering Class Members

- Methods That Return Information About One Specific Member

```
Field getField(String name)
Field getDeclaredField(String name)
Method getMethod(String name)
Method getDeclaredMethod(String name)
Constructor getConstructor(String name)
Constructor getDeclaredConstructor(String name)
```

The “Class” Class

The `Class` Class

- Browsing the Java 2 SE API you will find the `Class` class. A class that its name is “Class”.
  http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Class.html
- Each one of the values that exist during our code execution is either a primitive type value (e.g. int, double, float, byte etc.) or a class type reference (any reference for any object is a class type reference).
- For every class type reference, the JVM maintains an immutable instance of the class `java.lang.Class`.

The `Class` Class

- All objects extend (either directly or indirectly) from the `Object` class. Therefore, any of the methods that were declared within the `Object` class can be called on every object of every type.
- One of the methods we can find in `Object` is `getClass()`.

  http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Object.html
- For any object on which we call the `getClass()` method we will get a reference for an object of type `Class` that describes the class type of our object (the object on which we called the `getClass()` method).

This is just one of the ways through which we can get a reference for an object of type Class that describes a class.

The `Class` Class

- Getting a reference for a `Class` object is the first step for using any of the reflection capabilities.
- Once we get the reference for a `Class` object we can call any of the methods that were declared in `Class` on that object.
- Various methods that were declared in `Class` allow us getting detailed information about all members (methods, variables and constructors) of that class. Including the private ones.
- The methods and classes that belong to the Reflection API even provide a way to indirectly call methods on the object we currently inspect.

None of the classes that belong to java.lang.reflect has a public constructor allowing us to instantiate it. The only way to get objects of these various classes is first getting the relevant Class object and then invoke various methods (methods that were declared in Class class) on that object and get as their replies references for various objects instantiated from various classes that belong to the Reflection API (e.g. Calling the getDeclaredMethods() method on object of type Class that describes a specific class type will return an array that each one of its components is a reference for an object of type Method that describes a specific method in the class the Class object describes).

Using the classes that are part of the Reflection API we can even indirectly call a method on the object we inspect (as when using functions pointers in C++).

Retrieving a `Class` Object

- Getting a reference for a `Class` object is the first step for using any of the reflection capabilities.
- Once we get the reference for a `Class` object we can call any of the methods that were declared in `Class` on that object.
- Various methods that were declared in `Class` allow us getting detailed information about all members (methods, variables and constructors) of that class. Including the private ones.
- There are several ways for retrieving a `Class` object that describes a class we are interested at. The following slides describe these ways.

The `Object.getClass()` Method

- The `getClass()` method, that was declared in `Object`, can be invoked on every object.
- When calling this method on a specific object we get a reference for a Class object that describes the class from which that specific object was instantiated.

```
String str = new String("abc");
Class strClass = str.getClass();
```

- Calling the `getClass()` method we can get a `Class` object that describes a class from which an object was instantiated only.

The `Object.getClass()` Method

- We can't use this technique to get a `Class` object that describes an interface.
- We can use this technique to retrieve a `Class` object that describes a class that so far it was less clear for us that our object was instantiated from.

```
int []vec = new int[10];
Class vecClass = vec.getClass();
```


By running the following code you can get the name of the class from which an array of int is instantiated.

```
import java.util.*;


public class VecClass
{
public static void main(String args[])
{
int[] vec = new int[10];
Class vecClass = vec.getClass();
System.out.println("vecClass holds a reference for
an object that was instantiated from "+vecClass.getName());
}
}
```

You can get this code from this course server. The filename is VecClass.java.

The `.class` Syntax

- Appending `.class` to the name of a class type returns a reference to the Class object that describes the class type we used its name.

```
Class stringClass = String.class;
```

- You can use this special syntax to get even the Class object that describes an interface class type.

```
Class cloneableClass = Cloneable.class;
```


By running the following code you can get the name of the Cloneable interface class type.

```
import java.util.*;


public class GettingClassReferenceSample2
{
    public static void main(String args[])
    {
        Class vecClass = Cloneable.class;
        System.out.println("The Cloneable class type is " +
            vecClass.getName());
    }
}
```

You can get this code from this course server. The filename is GettingClassReferenceSample2.java.

The `Class.forName()` Method

- The `Class` class includes the `forName()` static method. Calling this method sending a full qualified name of a class type returns a reference to an object of type Class that describes the class type.

```
Class stringClass = Class.forName("java.lang.String");
```

Primitive Types

- Each primitive type (e.g. double, float etc..) has a class that represents it at runtime.
- These classes, that represent the primitive types, are separated from the wrapper classes (e.g. `Double`, `Float` etc..).
- While the wrapper classes are used to bridge between the primitive types world and the object world, the primitive types' classes are used to represent the primitive type values' types within the running code.

Primitive Types

- One of the possible ways for getting a `Class` object that represents a primitive type's class is by using the ".class" syntax. Writing `.class` preceding with the primitive type name will provide us with a reference to a `Class` object that represents the primitive type's class.

```
Class doubleClass = double.class;
```

- An alternative way can be using the `TYPE` static field we can find in each one of the wrapper classes.

```
Class doubleClass = Double.TYPE;
```

12/02/10 © Abelski eLearning 11

Running the following code will print out to the screen the names of some of the primitive types you know. You can find this file on our course server (file name is: PrimitiveTypes.java).

```
import java.util.*;

public class PrimitiveTypes
{
public static void main(String args[])
{
Class intClass = int.class;
Class floatClass = float.class;
Class doubleClass = double.class;
Class booleanClass = boolean.class;
System.out.println("int class name is "+intClass.getName());
System.out.println("float class name is "
+floatClass.getName());
System.out.println("double class name is"
+doubleClass.getName());
System.out.println("boolean class name is "
+booleanClass.getName());
}
}
```

Class Methods

- The `Class` class includes several methods that when we call them we get a reference to object of type Class.

`Class.getSuperclass()`

Calling this method returns a reference to an object of type `Class` that describes the super class.

`Class.getClasses()`

Calling this methods returns an array of references to objects of type `Class` representing all the public classes and interfaces that are members of the class represented by this `Class` object.

Try to run the following code. Note the printing out. You can find this code ready for download on our course's server. The file name is ClassMethods.

```
import java.util.*;

public class ClassMethods
{
public static void main(String args[])
{
Class rectangleClass = Rectangle.class;
Class rectangleSuperClass =
rectangleClass.getSuperclass();
Class classes[] = rectangleClass.getClasses();
Rectangle.StrongRectangle recy = new
Rectangle().new StrongRectangle();
Class recyClass = recy.getClass();
Class enclosingClass =
recyClass.getEnclosingClass() ;
System.out.println("rectangle class name is"
+rectangleClass.getName());
```

Class Methods

`Class.getDeclaredClasses()`

Calling this methods returns an array of references to objects of type `Class` representing all classes and interfaces that are members of the class represented by this `Class` object.

`Class.getEnclosingClass()`

Calling this methods returns a reference to object of type `Class` representing the immediately enclosing class.


```
System.out.println("rectangle super class name is "
                    +rectangleSuperClass);
System.out.println("rectangle memebers classes
                    names are");
for(int i=0; i<classes.length; i++)
                    {
System.out.println(classes[i].getName());
                    }
System.out.println("strong rectangle enclosing class is"
                    +enclosingClass.getName());
                    }
                    }

                    class Shape {}

                    class Rectangle extends Shape
                    {
                    String name;
                    private Date creationDate;
                    double width;
                    double height;
                    class StrongRectangle { }
                    }
```

Method,Constructor & Field

- The `Method`, `Constructor` & `Field` classes represent class members.
- All of these classes include the following method:

  `Class.getDeclaringClass()`

  Calling this method returns a reference to an object of type `Class` that describes the class this member belongs to.

Class Modifiers & Types

- A class can be declared with modifiers. The possible modifiers include the following:

  Access Modifiers – public, private & protected

  Abstract Modifier – abstract

  Static Modifier – static (you can declare an inner class as abstract)

  Final Modifier – final

  Strict Floating Point Behavior – strictfp

- The `Class` class includes the method `getModifiers()`. Calling that method on a Class object returns an integer number that each one of its bits (turned on/off) tells something about the class modifiers

strictfp is a Java keyword used to mark classes, methods and interfaces so that every float or double value in their variables will be made in the IEEE, as it used to be in the older versions of Java by default.

Class Modifiers & Types

- The class Modifier includes methods that can assist decoding `getModifiers()` returned value. This class also includes final fields for each one of the possible modifiers values.
  http://java.sun.com/j2se/1.5.0/docs/api/java/lang/reflect/Modifier.html

```
Class stringClass = String.class;
System.out.println(Modifier.toString(stringClass.getModifiers()));
```


The following code presents simple usage of the Class.getModifiers() method together with Modifier.toString() that decode the value getModifiers returns. You can find this sample code on our course's server. The filename is ClassModifiers.java.

```
import java.util.*;
import java.lang.reflect.*;

public class ClassModifiers
{
public static void main(String args[])
{
Class stringClass = String.class;
System.out.println(Modifier.
toString(stringClass.getModifiers()));
}
}
```

Discovering Class Members

- Once we retrieve a `Class` object that describes the class we want to analyze, all that is left is calling various methods on that object in order to get objects that describe the class members.
- The class members are described by objects instantiated from the following classes: `Method`, `Field` & `Constructor`.
- Once we retrieve an object that describes one of our class members, we can call various methods on that specific object in order to get more information on that member.

The following code sample demonstrate using these methods. You can get the source code file on our course's server. The file name is ReflectionDemo.java.

```
import java.lang.reflect.*;


public class ReflectionDemo
{
private static String name;
private static int blanks;
public static void main(String[] args)
{

if (args.length!=1)
{
System.out.println("\nERROR: You should give a
name of ONE class !");
return;
}

name = args[0];

Class classObj;
```

Discovering Class Members

- Methods That Return List of Public Members Only

  (Excluding The Inherited Ones)

  Field[] getFields()

  Method[] getMethods()

  Constructor[] getConstructors()

- Methods That Return List of All Members

  (Excluding The Inherited Ones)

  Field[] getDeclaredFields()

  Method[] getDeclaredMethods()

  Constructor[] getDeclaredConstructors()


```
try
{
classObj = Class.forName(name);
System.out.println("Report fo " + name + " : \n");
if (!printSuperClasses(classObj))
System.out.println("This classObj doesn't" +
"extend any other classObj !");
printConstructors(classObj);
printMethods(classObj);
printFields(classObj);
}
catch(ClassNotFoundException e)
{
System.out.println("Class not found.");
}
}

public static void printBlanks()
{
for (int i=0; i< blanks; i++)
System.out.print(" ");
}
```

Discovering Class Members

- Methods That Return Information About One Specific Member

```
Field getField(String name)
Field getDeclaredField(String name)
Method getMethod(String name)
Method getDeclaredMethod(String name)
Constructor getConstructor(String name)
Constructor getDeclaredConstructor(String name)
```