The Class Members

Fields, Methods & Constructors

- Class members (fields, methods & constructors) are represented using objects instantiated from the following classes:

  Method

  http://java.sun.com/javase/6/docs/api/java/lang/reflect/Method.html

  Field

  http://java.sun.com/javase/6/docs/api/java/lang/reflect/Field.html

  Constructor

  http://java.sun.com/javase/6/docs/api/java/lang/reflect/Constructor.html

Fields

- An object of type Field represents a field declared in a class.
- The methods declared in java.lang.reflect.Field can provide more information about the field, such as its name, type, modifiers & annotations.

Sample Code

```
Fields[] fields = String.class.getDeclaredFields();
for(int i=0; i<fields.length; i++)
{
    System.out.println(fields[i].getName());
    System.out.println(Modifier.toString(fields[i].getModifiers());
}
```

Fields

- The Field.set() method allows changing the value in the represented field in a specific object.

  public void set(Object obj, Object value)

  The first argument is a reference to the object that we want to change its field value.

  The second argument represents the value we want to set.

  Sample Code

```
Rectangle rec = new Rectangle(200,300);
Field field = rec.class.getDeclaredField("width");
field.set(rec,new Double(140));
```

Methods

- An object of type Method represents a method in a class.
- The methods declared in java.lang.reflect.Method can provide more information about the method, such as its name, returned type, parameters, modifiers & annotations.

Sample Code

```
Method[] methods = String.class.getDeclaredMethods();
for(int i=0; i<methods.length; i++)
{
        System.out.println(methods[i].getName());
        System.out.println(mehtods[i].getReturnType());
}
```

Methods

- The Method.invoke() method allows invoking the represented method on an object we choose.

  public Object invoke(Object obj, Object... args)

  The first argument is a reference to the object on which we want to invoke the method.

  The rest of the arguments represent the values we want to send to the invoked method.

  Sample Code

```
Rectangle rec = new Rectangle(200,300);
Method setMethod = recClass.getDeclaredMethod("set",double.class,double.class);
setMethod.invoke(rec,new Double(5), new Double(4));
```

Constructors

- An object of type Constructor represents a constructor in a class.
- The methods declared in java.lang.reflect.Constructor can provide more information about the constructor, such as its name, parameters, modifiers & annotations.

Sample Code

```
Rectangle rec = new Rectangle();
Class recClass = rec.getClass();
Constructor constructors[] = recClass.getDeclaredConstructors();
for(int i=0; i< constructors.length; i++)
        System.out.println(i+". "+constructors[i].getName()
        +":"+Modifier.toString(constructors[i].getModifiers()));
```

Class Instantiation

- The Class.newInstance() method enables us to instantiate a class without using the “new” operator.
- Using the newInstance() method we can write a code that instantiate a class that is not known when compiling the code.

Sample Code

```
Class instantiatedClass = Class.forName("Rectangle");
Object ob = instantiatedClass.newInstance();
System.out.println(ob);
```

The Class Members

Fields, Methods & Constructors

- Class members (fields, methods & constructors) are represented using objects instantiated from the following classes:

  Method

  http://java.sun.com/javase/6/docs/api/java/lang/reflect/Method.html

  Field

  http://java.sun.com/javase/6/docs/api/java/lang/reflect/Field.html

  Constructor

  http://java.sun.com/javase/6/docs/api/java/lang/reflect/Constructor.html

Fields

- An object of type Field represents a field declared in a class.
- The methods declared in java.lang.reflect.Field can provide more information about the field, such as its name, type, modifiers & annotations.

Sample Code

```
Fields[] fields = String.class.getDeclaredFields();
for(int i=0; i<fields.length; i++)
{
    System.out.println(fields[i].getName());
    System.out.println(Modifier.toString(fields[i].getModifiers());
}
```


We recommend you to go over the ReflectionDemo code for more demonstration for getting more information about fields.

Fields

- The Field.set() method allows changing the value in the represented field in a specific object.

  public void set(Object obj, Object value)

  The first argument is a reference to the object that we want to change its field value.

  The second argument represents the value we want to set.

  Sample Code

  Rectangle rec = new Rectangle(200,300);

  Field field = rec.class.getDeclaredField(“width”);

  field.set(rec,new Double(140));

The following code is a small sample for accessing a field value and changing it. The code is available on our course server. The filename is FieldValueSet.java.

```
import java.util.*;
import java.lang.reflect.*;
public class FieldValueSet
{
	public static void main(String args[])
	{
		Rectangle rec = new Rectangle();
		System.out.println("area before = " + rec.area());
		Class recClass = rec.getClass();
		try
		{
			Field widthField =
				recClass.getDeclaredField("width");
			widthField.set(rec,new Double(5));
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
	}
}
```

```
class Rectangle
{
	double width;
	double height;
	Rectangle()
	{
		width=10;
		height=10;
	}
	double area()
	{
		return width*height;
	}
}
```

Methods

- An object of type Method represents a method in a class.
- The methods declared in java.lang.reflect.Method can provide more information about the method, such as its name, returned type, parameters, modifiers & annotations.

Sample Code

```
Method[] methods = String.class.getDeclaredMethods();
for(int i=0; i<methods.length; i++)
{
    System.out.println(methods[i].getName());
    System.out.println(mehtods[i].getReturnType());
}
```


We recommend you to go over the ReflectionDemo code for more demonstration for getting more information about methods.

```
class Rectangle
{
	double width;
	double height;
	Rectangle()
	{
		width=10;
		height=10;
	}
	double area()
	{
		return width*height;
	}
	void set(double wVal, double hVal)
	{
		width = wVal;
		height = hVal;
	}
}
```

Methods

- The Method.invoke() method allows invoking the represented method on an object we choose.

  public Object invoke(Object obj, Object... args)

  The first argument is a reference to the object on which we want to invoke the method. The rest of the arguments represent the values we want to send to the invoked method.

  Sample Code

  Rectangle rec = new Rectangle(200,300);

  Method setMethod = recClass.getDeclaredMethod("set",double.class,double.class);

  setMethod.invoke(rec,new Double(5), new Double(4));

Check the following code (you can download it from our course server. Its file name is MethodCall.java).

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

public class MethodCall
{
	public static void main(String args[])
	{
		Rectangle rec = new Rectangle();
		System.out.println("area before = " + rec.area());
		Class recClass = rec.getClass();
		try
		{
Method setMethod =
recClass.getDeclaredMethod("set",double.class,double.class);
setMethod.invoke(rec,new Double(5), new Double(4));
		}
		catch(Exception e) {e.printStackTrace();}
		System.out.println("area before = " + rec.area());
	}
}
```

Constructors

- An object of type Constructor represents a constructor in a class.
- The methods declared in java.lang.reflect.Constructor can provide more information about the constructor, such as its name, parameters, modifiers & annotations.

Sample Code

```
Rectangle rec = new Rectangle();
Class recClass = rec.getClass();
Constructor constructors[] = recClass.getDeclaredConstructors();
for(int i=0; i< constructors.length; i++)
    System.out.println(i+". "+constructors[i].getName()
    +":"+Modifier.toString(constructors[i].getModifiers()));
```

09/10/09 © Abelski eLearning 9

In addition, please find below an additional sample code you can download from our course's server. Filename is ConstructorReflection.java. More demonstrations can be found in ReflectionDemo (from the previous slides).

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

public class ConstructorReflection
{
        public static void main(String args[])
        {
                Rectangle rec = new Rectangle();
                Class recClass = rec.getClass();
                Constructor constructors[] =
                        recClass.getDeclaredConstructors();
                for(int i=0; i< constructors.length; i++)
                {
                        System.out.println(i+". "+constructors[i].
                        getName()+":"+Modifier.toString(
                        constructors[i].getModifiers()));
                }
        }
}
```

Class Instantiation

- The Class.newInstance() method enables us to instantiate a class without using the “new” operator.
- Using the newInstance() method we can write a code that instantiate a class that is not known when compiling the code.

Sample Code

```
Class instantiatedClass = Class.forName("Rectangle");
Object ob = instantiatedClass.newInstance();
System.out.println(ob);
```

09/10/09 © Abelski eLearning 10

Check the following code for demonstration of instantiating a class using the newInstance() method.

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

public class ClassInstantiation
{
	public static void main(String args[])
	{
		try
		{
			Class instantiatedClass =
Class.forName("Rectangle");
			Object ob = instantiatedClass.newInstance();
			System.out.println(ob);
		}
		catch(Exception e) {}
	}
}
```

```
class Rectangle
{
	double width;
	double height;
	protected Rectangle()
	{
		width=10;
		height=10;
	}
	public Rectangle(double hVal, double wVal)
	{
		width = wVal;
		height = hVal;
	}
	public String toString()
	{
		return "width="+width+" height="+height+" area="+
(width*height);
	}
}
```