Reflection

Bob Singh

Reflection

- The reflection API reflects the classes, interfaces, and objects in the current JVM.
- With the reflection API, you can
  - Determine the class of an object.
  - Class’s modifiers, fields, methods, constructors, and superclasses.
  - Create an instance of a class whose name is not known until runtime.
  - What constants and method declarations belong to an interface.
  - Get and set the value of an object's field, even if the field name is unknown to your program until runtime.
  - Invoke a method on an object, even if the method is not known until runtime.

Reflection API

- **`java.lang.Class`**
- **`java.lang.reflect.AccessibleObject`**
  - **`java.lang.reflect.Field`**
  - **`java.lang.reflect.Method`**
  - **`java.lang.reflect.Constructor`**
- **`java.lang.reflect.Array`**
- **`java.lang.reflect.Modifier`**
- **`java.lang.InvocationtargetException`**
- **`java.lang.UndeclaredThrowableException`**
- **`java.lang.ReflectPermission`**

Class

- Reflects classes and interfaces
- To get class and superclass information:

```
Button b = new Button();
Class c = b.getClass();
Class s = c.getSuperclass();
```

Class

```
import java.lang.reflect.*;
class name
{
    public static void main(String[] args) {
    String s = new String();
    Class c = s.getClass();
    Class su = c.getSuperClass();
    System.out.println("Subclass:" +
  c.getName();
    System.out.println("Superclass:"+
  su.getName();
}}
```

```
Output: Subclass:java.lang.String
        Superclass:java.lang.Object
```

Class Modifier

- Class declaration may include the modifiers:
  - **`public`**
  - **`abstract`**
  - **`final`**
- To identify the modifiers of a class at runtime:
  - Invoke **`getModifiers`** on a **`Class`** object to retrieve a set of modifiers.
  - Check the modifiers by calling **`isPublic`**, i**`sAbstract`** and **`isFinal`**.

Class Modifier

```
import java.lang.reflect.*;
class modifier
{ public static void main(String[] args)
{ String s = new String();
  Class c = s.getClass();
  int m = c.getModifiers();
  if (Modifier.isPublic(m))
     System.out.println("public");
  if (Modifier.isAbstract(m))
     System.out.println("abstract");
  if (Modifier.isFinal(m))
     System.out.println("final");
} }
```

```
Output: public
        final
```

Class Fields

- Represents fields of a class.
- Has
  - **`getFields()`** method, returns an array of Fields
  - **`getType()`** method to get the data type of the field
  - **`getName()`** to get the name of the field

Class Fields

```
import java.lang.reflect.*;
class field {
public static void main(String[] args) {
Integer i = new Integer(0);
Class c = i.getClass();
Field[] publicFields = c.getFields();
for (int i = 0; i < publicFields.length; i++){
   String fieldName = publicFields[i].getName();
   Class typeClass = publicFields[i].getType();
   String fieldType = typeClass.getName();
   System.out.println("Name: " + fieldName + ",
  Type: " + fieldType); }}}
```

```
Output:   Name: MIN_VALUE, Type: int
          Name: MAX_VALUE, Type: int
          Name: TYPE, Type: java.lang.Class
```

Class Constructors

- To get class's constructors, you can invoke the **`getConstructors`** method, which returns an array of **`Constructor`** objects.
- **`Constructor`** class determines the constructor's name, set of modifiers, parameter types, and set of throwable exceptions.
- Create a new instance of the **`Constructor`** object's class with the C**`onstructor.newInstance`** method.

Class Constructors

```
import java.lang.reflect.*;
class constructor {
public static void main(String[] args) {
Integer i = new Integer(0);
Class c = i.getClass();
Constructor[] theConstructors = c.getConstructors();
for (int j = 0; j < theConstructors.length; j++)
{
  System.out.print("( ");
  Class[] parameterTypes =
         theConstructors[j].getParameterTypes();
  for (int k = 0; k < parameterTypes.length; k ++) {
         String parameterString =
   parameterTypes[k].getName();
     System.out.print(parameterString + " ");}
         System.out.println(")"); } }}
```

```
Output:( java.lang.String )
        ( int )
```

Method

- To find out public methods belonging to a class, invoke the method named **`getMethods;`** returns an array containing **`Method`** objects.
  - Method's name
  - Return type
  - Parameter types
  - Set of modifiers
  - Set of throwable exceptions.
- With **`Method.invoke`**, you can call the method itself.

Method

```
import java.lang.reflect.*;
class method { public static void main(String[] args) {
 String s = new String(); Class c = s.getClass();
 Method[] m = c.getMethods();
 for (int i = 0; i < m.length; i++) {
   System.out.println("Name: " + m[i].getName());
   String returnString =  m[i].getReturnType().getName();
   System.out.println("Return Type: " + returnString);
   Class[] p = m[i].getParameterTypes();
   System.out.print("Parameter Types: ");
   for (int k = 0; k < p.length; k ++) {
    String parameterString = p[k].getName();
    System.out.print(" " + parameterString);}
    System.out.println();}}}
   //Outputs the method names with  return type and
  //parameter names
```

Interfaces implemented by a Class

- **`getInterfaces`** method determines which interfaces a class implements.
  - The **`getInterfaces`** method returns an array of **`Class`** objects.
  - You can invoke the **`getName`** method on the **`Class`** objects to retrieve the interface names.

Interfaces

```
import java.lang.reflect.*;
import java.io.*;
class InterfaceEx {
public static void main(String[] args) {
   String s = new String();
  Class c = s.getClass();
  Class[] interfaces = c.getInterfaces();
 if ((interfaces != null) && (interfaces.length > 0)) {
      if (c.isInterface())
              System.out.println(" extends ");
      else
              System.out.print(" implements ");
for(int i = 0; i < interfaces.length; i++) {
   if (i > 0) System.out.print(", ");
        System.out.print(interfaces[i].getName());}}}}
//Output  implements java.io.Serializable,
  java.lang.Comparable, java.lang.CharSequence
```