

```
001 // The Java Reflection API is used to manipulate classes
002 // and everything in a class including fields, methods,
003 // constructors, private data, etc.
004
005 // Because using the Reflection API is most often Dynamic
006 // it can slow down a program because the JVM can't
007 // optimize the code.
008
009 // The Reflection API can't be used with applets because
010 // of the added security applets require.
011
012 // Because this API allows you to do things like access
013 // private fields, methods, etc. it should be used
014 // sparingly, or else potentially destroy the logic
015 // of a program
016
017 import java.lang.reflect.Constructor;
018 import java.lang.reflect.Field;
019 import java.lang.reflect.InvocationTargetException;
020 import java.lang.reflect.Method;
021 import java.lang.reflect.Modifier;
022
023 public class TestingReflection {
024
025     public static void main(String[] args){
026
027         // Getting the class Object for UFOEnemyShip
028         // Everything in Java has a Class Object
029
030         Class reflectClass = UFOEnemyShip.class;
031
032         // Get the class name of an Object
033
034         String className = reflectClass.getName();
035
036         System.out.println(className + "\n");
037
038         // Check modifiers of a class
039         // isAbstract, isFinal, isInterface, isPrivate, isProtected,
040         // isStatic, isStrict, isSynchronized, isVolatile
041
042         int classModifiers = reflectClass.getModifiers();
043
044         System.out.println(Modifier.isPublic(classModifiers) + "\n");
045
046         // You can get a list of interfaces used by a class
047         // Class[] interfaces = reflectClass.getInterfaces();
048
049         // Get the super class for UFOEnemyShip
050
051         Class classSuper = reflectClass.getSuperclass();
052
053         System.out.println(classSuper.getName() + "\n");
054
055         // Get the objects methods, return type and parameter type
056
```

```
057 Method[] classMethods = reflectClass.getMethods();
058
059 for(Method method : classMethods){
060     // Get the method name
061     System.out.println("Method Name: " + method.getName());
062
063     // Check if a method is a getter or setter
064     if(method.getName().startsWith("get")) {
065         System.out.println("Getter Method");
066     } else if(method.getName().startsWith("set")) {
067         System.out.println("Setter Method");
068     }
069
070     // Get the methods return type
071     System.out.println("Return Type: " + method.getReturnType());
072
073     Class[] parameterType = method.getParameterTypes();
074
075     // List parameters for a method
076     System.out.println("Parameters");
077     for(Class parameter : parameterType){
078         System.out.println(parameter.getName());
079     }
080
081     System.out.println();
082 }
083
084 // How to access class constructors
085 Constructor constructor = null;
086
087 Object constructor2 = null;
088
089 try {
090     // If you know the parameters of the constructor you
091     // want you do the following.
092
093     // To return an array of constructors instead do this
094     // Constructor[] constructors = reflectClass.getConstructors();
095
096     // If the constructor receives a String you'd use the
097     // parameter new Class[]{String.class}
098     // For others use int.class, double.class, etc.
```

```
114
115         constructor = reflectClass.getConstructor(new Class[]
{EnemyShipFactory.class});
116
117         // Call a constructor by passing parameters to create an object
118
119         constructor2 = reflectClass.getConstructor(int.class,
String.class).newInstance(12, "Random String");
120     }
121
122     catch (NoSuchMethodException | SecurityException e) {
123         // Exceptions thrown
124         e.printStackTrace();
125     } catch (InstantiationException e) {
126         // TODO Auto-generated catch block
127         e.printStackTrace();
128     } catch (IllegalAccessException e) {
129         // TODO Auto-generated catch block
130         e.printStackTrace();
131     } catch (IllegalArgumentException e) {
132         // TODO Auto-generated catch block
133         e.printStackTrace();
134     } catch (InvocationTargetException e) {
135         // TODO Auto-generated catch block
136         e.printStackTrace();
137     }
138
139     // Return the parameters for a constructor
140
141     Class[] constructParameters = constructor.getParameterTypes();
142
143     for(Class parameter : constructParameters){
144
145         System.out.println(parameter.getName());
146
147     }
148
149     UFOEnemyShip newEnemyShip = null;
150
151     EnemyShipFactory shipFactory = null;
152
153     try {
154
155         // Create a UFOEnemyShip object by calling newInstance
156
157         newEnemyShip = (UFOEnemyShip)
constructor.newInstance(shipFactory);
158
159     }
160
161     catch (InstantiationException | IllegalAccessException |
IllegalArgumentException | InvocationTargetException e) {
162
163         e.printStackTrace();
164
165     }
166
```

```
167 // Now I can call methods in the UFOEnemyShip Object
168
169 newEnemyShip.setName("Xt-1000");
170 System.out.println("EnemyShip Name: " + newEnemyShip.getName());
171
172 // Access private fields using reflection
173
174 // Field stores info on a single field of a class
175
176 Field privateStringName = null;
177
178 try {
179
180     // Create a UFOEnemyShip object
181
182     UFOEnemyShip enemyshipPrivate = new UFOEnemyShip(shipFactory);
183
184     // Define the private field you want to access
185     // I can access any field with just its name dynamically
186
187     privateStringName =
123 UFOEnemyShip.class.getDeclaredField("idCode");
188
189     // Shuts down security allowing you to access private fields
190
191     privateStringName.setAccessible(true);
192
193     // Get the value of a field and store it in a String
194
195     String valueOfName = (String)
123 privateStringName.get(enemyshipPrivate);
196
197     System.out.println("EnemyShip Private Name: " + valueOfName);
198
199     // Get access to a private method
200     // getDeclaredMethod("methodName", methodParameters or null)
201
202     // Since I provide the method name as a String I can run any
123 method
203     // without needing to follow the normal convention methodName()
204
205     String methodName = "getPrivate";
206
207     Method privateMethod =
123 UFOEnemyShip.class.getDeclaredMethod(methodName, null);
208
209     // Shuts down security allowing you to access private methods
210
211     privateMethod.setAccessible(true);
212
213     // get the return value from the method
214
215     String privateReturnVal = (String)
123 privateMethod.invoke(enemyshipPrivate, null);
216
217     System.out.println("EnemyShip Private Method: " +
123 privateReturnVal);
```

```
218
219         // Execute a method that has parameters
220
221         // Define the parameters expected by the private method
222
223         Class[] methodParameters = new Class[]{Integer.TYPE,
String.class};
224
225         // Provide the parameters above with values
226
227         Object[] params = new Object[]{new Integer(10), new
String("Random")};
228
229         // Get the method by providing its name and a Class array with
parameters
230
231         privateMethod =
UFOEnemyShip.class.getDeclaredMethod("getOtherPrivate", methodParameters);
232
233         // Shuts down security allowing you to access private methods
234
235         privateMethod.setAccessible(true);
236
237         // Execute the method and pass parameter values. The return
value is stored
238
239         privateReturnVal = (String)
privateMethod.invoke(enemyShipPrivate, params);
240
241         System.out.println("EnemyShip Other Private Method: " +
privateReturnVal);
242
243     }
244
245     catch (NoSuchFieldException | SecurityException e) {
246         // TODO Auto-generated catch block
247         e.printStackTrace();
248     }
249
250     catch (IllegalArgumentException e) {
251         // TODO Auto-generated catch block
252         e.printStackTrace();
253     }
254
255     catch (IllegalAccessException e) {
256         // TODO Auto-generated catch block
257         e.printStackTrace();
258     }
259
260     catch (NoSuchMethodException e) {
261         // TODO Auto-generated catch block
262         e.printStackTrace();
263     }
264
265     catch (InvocationTargetException e) {
266         // TODO Auto-generated catch block
267         e.printStackTrace();
```

```
268         }  
269  
270     }  
271  
272 }
```