```java
// The Java Reflection API is used to manipulate classes
// and everything in a class including fields, methods,
// constructors, private data, etc.

// Because using the Reflection API is most often Dynamic
// it can slow down a program because the JVM can't
// optimize the code.

// The Reflection API can't be used with applets because
// of the added security applets require.

// Because this API allows you to do things like access
// private fields, methods, etc. it should be used
// sparingly, or else potentially destroy the logic
// of a program

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;

public class TestingReflection {

    public static void main(String[] args){

        // Getting the class Object for UFOEnemyShip
        // Everything in Java has a Class Object

        Class reflectClass = UFOEnemyShip.class;

        // Get the class name of an Object

        String className = reflectClass.getName();

        System.out.println(className + "\n");

        // Check modifiers of a class
        // isAbstract, isFinal, isInterface, isPrivate, isProtected,
        // isStatic, isStrict, isSynchronized, isVolatile

        int classModifiers = reflectClass.getModifiers();

        System.out.println(Modifier.isPublic(classModifiers) + "\n");

        // You can get a list of interfaces used by a class
        // Class[] interfaces = reflectClass.getInterfaces();

        // Get the super class for UFOEnemyShip

        Class classSuper = reflectClass.getSuperclass();

        System.out.println(classSuper.getName() + "\n");

        // Get the objects methods, return type and parameter type

        Method[] classMethods = reflectClass.getMethods();

        for(Method method : classMethods){

            // Get the method name

            System.out.println("Method Name: " + method.getName());

            // Check if a method is a getter or setter

            if(method.getName().startsWith("get")) {

                System.out.println("Getter Method");

            } else if(method.getName().startsWith("set")) {

                System.out.println("Setter Method");

            }

            // Get the methods return type

            System.out.println("Return Type: " + method.getReturnType());

            Class[] parameterType = method.getParameterTypes();

            // List parameters for a method

            System.out.println("Parameters");

            for(Class parameter : parameterType){

                System.out.println(parameter.getName());

            }

            System.out.println();

        }

        // How to access class constructors

        Constructor constructor = null;

        Object constructor2 = null;

        try {

            // If you know the parameters of the constructor you
            // want you do the following.

            // To return an array of constructors instead do this
            // Constructor[] constructors = reflectClass.getConstructors();

            // If the constructor receives a String you'd use the
            // parameter new Class[]{String.class}
            // For others use int.class, double.class, etc.
```

```java

            constructor = reflectClass.getConstructor(new Class[]{EnemyShipFactory.class});

            // Call a constructor by passing parameters to create an object

            constructor2 = reflectClass.getConstructor(int.class, String.class).newInstance(12, "Random String");
        }

        catch (NoSuchMethodException | SecurityException e) {
            // Exceptions thrown
            e.printStackTrace();
        } catch (InstantiationException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IllegalArgumentException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        // Return the parameters for a constructor

        Class[] constructParameters = constructor.getParameterTypes();

        for(Class parameter : constructParameters){

            System.out.println(parameter.getName());

        }

        UFOEnemyShip newEnemyShip = null;

        EnemyShipFactory shipFactory = null;

        try {

            // Create a UFOEnemyShip object by calling newInstance

            newEnemyShip = (UFOEnemyShip) constructor.newInstance(shipFactory);

        }

        catch (InstantiationException | IllegalAccessException | IllegalArgumentException | InvocationTargetException e) {

            e.printStackTrace();

        }

        // Now I can call methods in the UFOEnemyShip Object

        newEnemyShip.setName("Xt-1000");
        System.out.println("EnemyShip Name: " + newEnemyShip.getName());

        // Access private fields using reflection

        // Field stores info on a single field of a class

        Field privateStringName = null;

        try {

            // Create a UFOEnemyShip object

            UFOEnemyShip enemyshipPrivate = new UFOEnemyShip(shipFactory);

            // Define the private field you want to access
            // I can access any field with just its name dynamically

            privateStringName = 
UFOEnemyShip.class.getDeclaredField("idCode");

            // Shuts down security allowing you to access private fields

            privateStringName.setAccessible(true);

            // Get the value of a field and store it in a String

            String valueOfName = (String) 
privateStringName.get(enemyshipPrivate);

            System.out.println("EnemyShip Private Name: " + valueOfName);

            // Get access to a private method
            // getDeclaredMethod("methodName", methodParamters or null)

            // Since I provide the method name as a String I can run any 
method
            // without needing to follow the normal convention methodName()

            String methodName = "getPrivate";

            Method privateMethod = 
UFOEnemyShip.class.getDeclaredMethod(methodName, null);

            // Shuts down security allowing you to access private methods

            privateMethod.setAccessible(true);

            // get the return value from the method

            String privateReturnVal = (String) 
privateMethod.invoke(enemyshipPrivate, null);

            System.out.println("EnemyShip Private Method: " + 
privateReturnVal);
```

```java

            // Execute a method that has parameters

            // Define the parameters expected by the private method

            Class[] methodParameters = new Class[]{Integer.TYPE, String.class};

            // Provide the parameters above with values

            Object[] params = new Object[]{new Integer(10), new String("Random")};

            // Get the method by providing its name and a Class array with parameters

            privateMethod = UFOEnemyShip.class.getDeclaredMethod("getOtherPrivate", methodParameters);

            // Shuts down security allowing you to access private methods

            privateMethod.setAccessible(true);

            // Execute the method and pass parameter values. The return value is stored

            privateReturnVal = (String) privateMethod.invoke(enemyshipPrivate, params);

            System.out.println("EnemyShip Other Private Method: " + privateReturnVal);

        }

        catch (NoSuchFieldException | SecurityException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        catch (IllegalArgumentException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        catch (IllegalAccessException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        catch (NoSuchMethodException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        catch (InvocationTargetException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            }

        }

}
```