```
package lection1;

import java.lang.reflect.Modifier;

// Object Class Creation and General Operations
public class Test1 {
    public static void getClassObject() {
        // Literal
        Class<Double> cls0 = Double.class;
        System.out.println("Class: " + cls0); // Class: class
java.lang.Double

        Class<?> cls1 = Double.TYPE;
        System.out.println("Class: " + cls1); // Class: class
java.lang.Double

        // From Object
        Object obj = new java.util.Scanner(System.in);
        Class cls2 = obj.getClass();
        System.out.println("Class: " + cls2); // Class: class
java.util.Scanner

        // From String
        String name = "java.util.Scanner";
        Class<?> cls3 = null;
        try {
            cls3 = Class.forName(name);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        System.out.println("Class: " + cls3); // Class: class
java.util.Scanner
    }

    public static void showPreAnalysis() {
        Class cls = Double.class;

        System.out.println(cls.isInterface()); // false
        System.out.println(cls.isPrimitive()); // false
        System.out.println(cls.isArray());     // false
        Package p = cls.getPackage();
        System.out.println(p.getName());

        int mod = cls.getModifiers();
        System.out.println(Modifier.isPublic(mod));
        System.out.println(Modifier.toString(mod));

        System.out.println(cls.getName());
        System.out.println(cls.getSimpleName());

        System.out.println(cls.getSuperclass());
        for (Class c : cls.getInterfaces())
            System.out.println(c);
    }

    public static void simpleInstatntiate() {
        Class<String> cls = String.class;
        String o = null;
        try {
            o = cls.newInstance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
        System.out.println(o);
        // про cast придумать
    }

    public static void main(String[] args) {
        //Test1.getClassObject();
        //Test1.showPreAnalysis();
        Test1.simpleInstatntiate();
    }
}
```

```
package lection1;

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

public class Analysis1 {

    public static Class getTestClass() {
        return String.class;
    }

    public static Object create(Class cls) {
        Constructor[] ctors = cls.getDeclaredConstructors();
        AccessibleObject.setAccessible(ctors, true);
        for (int i = 0; i < ctors.length; i++) {
            System.out.println(i+"). " + ctors[i]);
        }

        //try to get desired constructor
        Constructor<String> ctor = null;
        try {
            ctor = cls.getDeclaredConstructor(char[].class);
        } catch (NoSuchMethodException e) {
            e.printStackTrace();
        }

        if (!Modifier.isPublic(ctor.getModifiers())) {
            ctor.setAccessible(true);
        }

        Object res = null;

        try {
            res = ctor.newInstance(new char[]{'a', 'b', 'c'});
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        }
        return res;
    }

    public static void withFields(Object obj) {
        Class cls = obj.getClass();
        Field[] flds = cls.getDeclaredFields();
        for (Field f : flds)
            System.out.println(f);
        AccessibleObject.setAccessible(flds, true);

        Field f = null;
        try {
            f = cls.getDeclaredField("value");
        } catch (NoSuchFieldException e) {
            e.printStackTrace();
        }

        f.setAccessible(true);
        try {
            System.out.println(f + " : " + Arrays.toString((char[])
f.get(obj)));
            //private final char[] java.lang.String.value : [a, b, c]
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }

        try {
            f.set(obj, new char[] {'d', 'e', 'f', 'g'});
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }

        try {
            System.out.println(f + " : " + Arrays.toString((char[])
f.get(obj)));
            //private final char[] java.lang.String.value : [d, e, f, g]
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
        try {
            Array.set(f.get(obj), 1, 'x');
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }

        try {
            System.out.println(f + " : " + Arrays.toString((char[])
f.get(obj)));
            //private final char[] java.lang.String.value : [d, x, f, g]
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }

    public static void withMethods(Object obj) {
        Method[] meths = obj.getClass().getDeclaredMethods();
        for (Method m : meths)
            System.out.println(m);
        AccessibleObject.setAccessible(meths, true);

        Method m = null;
        try {
            m = obj.getClass().getDeclaredMethod("length");
            System.out.println(obj + " length = " + m.invoke(obj)); // dxfg
length = 4
        } catch (NoSuchMethodException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }

        try {
            m = obj.getClass().getDeclaredMethod("substring", int.class,
int.class);
            System.out.println(obj + " substring = " + m.invoke(obj, 1,3));
// dxfg substring = xf
        } catch (NoSuchMethodException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }

        try {
            m = obj.getClass().getDeclaredMethod("valueOf", double.class);
            System.out.println("String.valueOf = " + m.invoke(null, 1.3)); //
String.valueOf = 1.3
        } catch (NoSuchMethodException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Object res = Analysis1.create(Analysis1.getTestClass());
        System.out.println(res);
        Analysis1.withFields(res);
        Analysis1.withMethods(res);
    }
}
```

```
package array;

import java.lang.reflect.Array;
import java.util.Arrays;

public class TestArray {

    public static void array1d() {
        Object arr = Array.newInstance(int.class, 5);
        System.out.println(arr.getClass().getName()); // [I
        System.out.println(arr.getClass().isArray()); // true
        System.out.println(arr.getClass().getComponentType()); // int
        System.out.println(Array.getLength(arr)); // 5
        for (int i = 0; i < Array.getLength(arr); i++)
            Array.set(arr, i, i*10);

        for (int i = 0; i < Array.getLength(arr); i++)
            System.out.print(Array.get(arr, i)+ ", ");
        System.out.println(); // 0, 10, 20, 30, 40,

        System.out.println(Arrays.toString((int[]) arr)); // [0, 10, 20, 30,
40]
    }

    public static void array2d() {
        Object matr = Array.newInstance(int.class, new int[] {3, 5});
        System.out.println(matr.getClass().getName()); // [[I
        System.out.println(matr.getClass().isArray()); // true
        System.out.println(matr.getClass().getComponentType()); // [I
        System.out.println(Array.getLength(matr)); // 3
        System.out.println(Array.getLength(Array.get(matr,0))); // 5

        for (int i = 0; i < Array.getLength(matr); i++) {
            for (int j = 0; j < Array.getLength(Array.get(matr, i)); j++) {
                Array.set(Array.get(matr, i), j, i + j);
            }
        }

        for (int i = 0; i < Array.getLength(matr); i++) {
            for (int j = 0; j < Array.getLength(Array.get(matr, i)); j++) {
                System.out.print(Array.get(Array.get(matr, i), j) + ", ");
            }
            System.out.println();
        }
    }


    public static void main(String[] args) {
        //TestArray.array1d();
        TestArray.array2d();
    }
}
```

```
package proxy1;

public interface Evaluatable {
    double evalf(double x);
}



package proxy1;

public class NumFunction implements Evaluatable {
    @Override
    public double evalf(double x) {
        return Math.sin(x)/x;
    }
}



package proxy1;

public class SimpleProxy implements Evaluatable {
    private NumFunction fun = null;

    public SimpleProxy(NumFunction fun) {
        this.fun = fun;
    }

    public SimpleProxy() {
        this(null);
    }

    public void setFun(NumFunction fun) {
        this.fun = fun;
    }

    private NumFunction getFun() {
        if (fun == null) {
            fun = new NumFunction();
        }
        return this.fun;
    }


    @Override
    public double evalf(double x) {
        System.out.println("x: " + x);
        double start = System.nanoTime();
        double res = fun.evalf(x);
        double finish = System.nanoTime();
        System.out.println("Elapsed time: " + (finish-start) + " ns");
        System.out.println("res: " + res);
        return res;
    }
}



package proxy1;

public class Main {
    public static void main(String[] args) {
        NumFunction f = new NumFunction();
        SimpleProxy p = new SimpleProxy(f);

        double x = 1.0;
        System.out.println("Function object: " + f.evalf(x));
        System.out.println("Function proxy: " + p.evalf(x));
    }
}
```

```
package proxy2;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

public class FunHandler implements InvocationHandler {
    private Object obj;

    public FunHandler(Object obj) {
        this.obj = obj;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args)
throws Throwable {
        for (Object a : args)
            System.out.println("arg: " + a);
        double start = System.nanoTime();
        double res = (double) method.invoke(obj, args);
        double finish = System.nanoTime();
        System.out.println("Elapsed time: " + (finish-start) + " ns");
        System.out.println("res: " + res);
        return res;
    }
}
```

```
package proxy2;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Proxy;

public class Main {
    public static void main(String[] args) {
        Class<?> proxyClass =
Proxy.getProxyClass(Evaluatable.class.getClassLoader(),
                new Class<?> [] {Evaluatable.class});
        System.out.println(proxyClass.getSimpleName());  // $Proxy0
        for (Constructor c : proxyClass.getDeclaredConstructors())
            System.out.println(c); // public
com.sun.proxy.$Proxy0(java.lang.reflect.InvocationHandler)
        ////
        NumFunction fun = new NumFunction();
        Class<?> proxy =
Proxy.getProxyClass(fun.getClass().getClassLoader(),
                fun.getClass().getInterfaces());
        System.out.println(proxy.getSimpleName());  // $Proxy0
        for (Constructor c : proxy.getDeclaredConstructors())
            System.out.println(c); // public
com.sun.proxy.$Proxy0(java.lang.reflect.InvocationHandler)

        fun.evalf(1.0);
        try {
            Evaluatable e = (Evaluatable)
proxy.getConstructor(InvocationHandler.class).newInstance(new
FunHandler(fun));
            //Evaluatable e = (Evaluatable) proxy.getConstructor(new
Class[] {InvocationHandler.class}).newInstance(new Object[] {new
FunHandler(fun)});
            e.evalf(1.0);
        } catch (InstantiationException ex) {
            ex.printStackTrace();
        } catch (IllegalAccessException ex) {
            ex.printStackTrace();
        } catch (InvocationTargetException ex) {
            ex.printStackTrace();
        } catch (NoSuchMethodException ex) {
            ex.printStackTrace();
        }

        ////
        Evaluatable e = (Evaluatable)
Proxy.newProxyInstance(fun.getClass().getClassLoader(),
                fun.getClass().getInterfaces(), new FunHandler(fun));
        e.evalf(1.0);
        fun.evalf(1.0);
    }
}
```