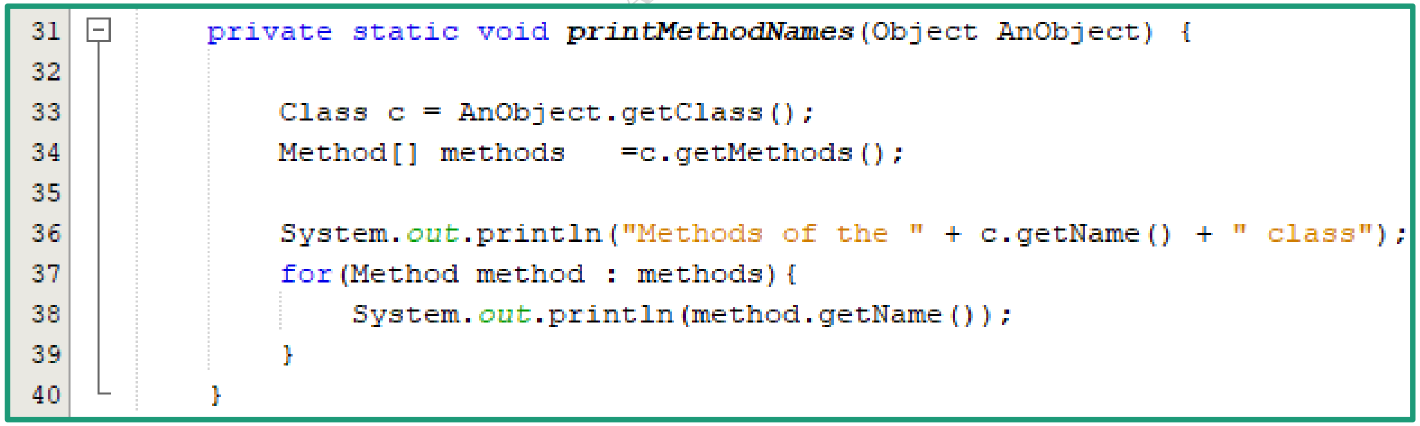
B.Sc. In Internet Systems Development.
Concurrent Programming.
Reflection.

LIMERICK INSTITUTE OF TECHNOLOGY
SCHOOL OF SCIENCE, ENGINEERING & I.T.
Department of Information Technology

Introduction

- Allows a program to inspect the contents of objects at runtime.
  - E.G. allows you to invoke arbitrary methods.
- You can inspect classes/interfaces/fields/methods (at runtime), without knowing the names of the classes/methods etc. at compile time.
- Very useful and powerful For Example:



```
    private static void printMethodNames(Object AnObject) {

        Class c = AnObject.getClass();
        Method[] methods   =c.getMethods();

        System.out.println("Methods of the " + c.getName() + " class");
        for(Method method : methods){
            System.out.println(method.getName());
        }
    }
```

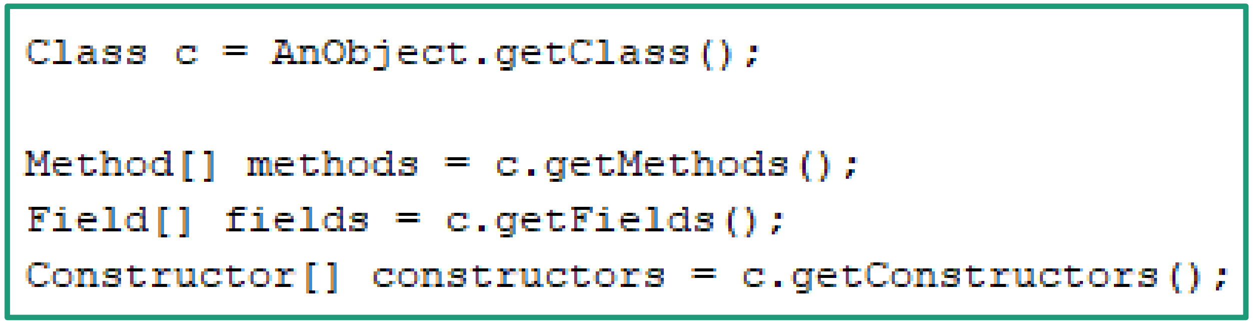
basics.Example1.java

Enumerating Class Methods

- Three classes underpin reflection (package java.lang.reflect).

  1. Field.
  2. Method.
  3. Constructor.

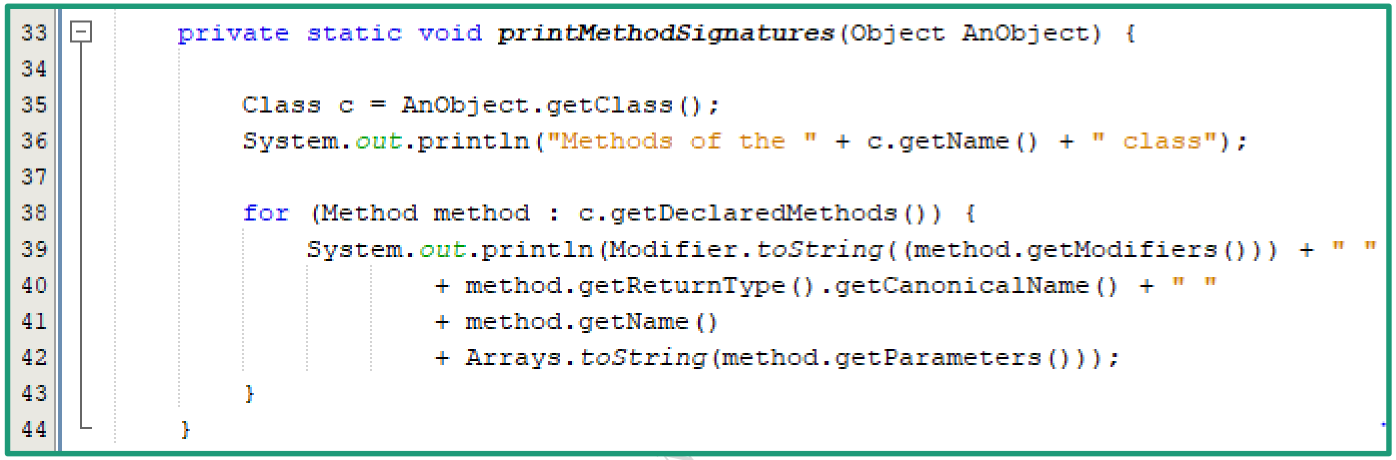
1. A fourth class (package java.lang), called class also plays a vital role.



```
Class c = AnObject.getClass();

Method[] methods = c.getMethods();
Field[] fields = c.getFields();
Constructor[] constructors = c.getConstructors();
```

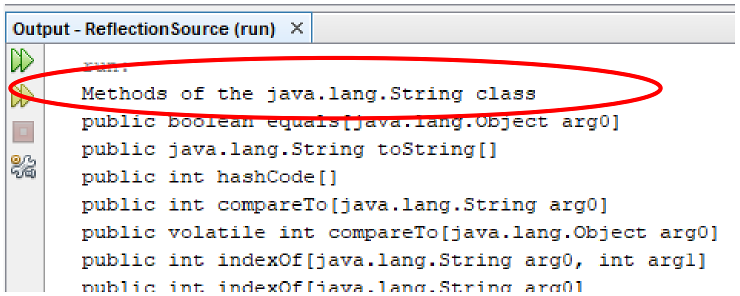
Enumerating The Methods of a Class

- Example2: Print method signatures:

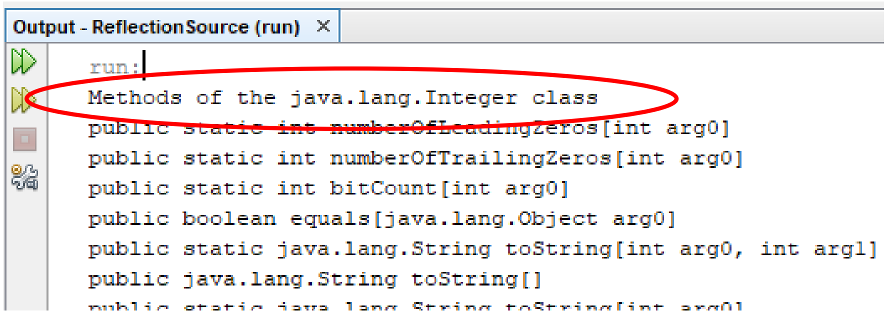


```java
    private static void printMethodSignatures(Object AnObject) {

        Class c = AnObject.getClass();
        System.out.println("Methods of the " + c.getName() + " class");

        for (Method method : c.getDeclaredMethods()) {
            System.out.println(Modifier.toString((method.getModifiers())) + " "
                    + method.getReturnType().getCanonicalName() + " "
                    + method.getName()
                    + Arrays.toString(method.getParameters()));
        }
    }
```

basics.Example2.java

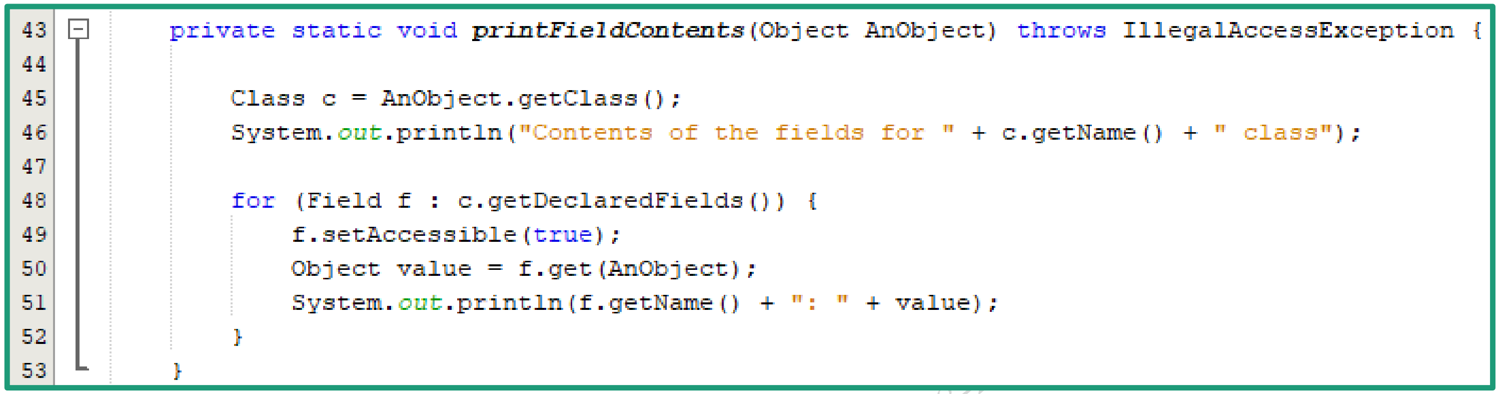


```
run:
Methods of the java.lang.String class
public boolean equals[java.lang.Object arg0]
public java.lang.String toString[]
public int hashCode[]
public int compareTo[java.lang.String arg0]
public volatile int compareTo[java.lang.Object arg0]
public int indexOf[java.lang.String arg0, int arg1]
public int indexOf[java.lang.String arg0]
```



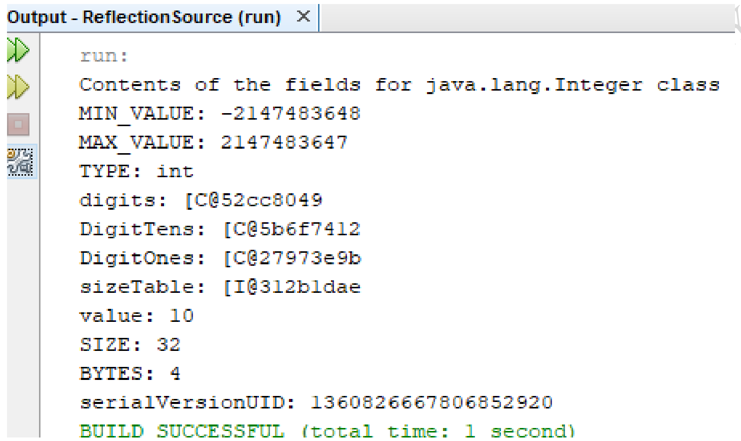
```
run:
Methods of the java.lang.Integer class
public static int numberOfLeadingZeros[int arg0]
public static int numberOfTrailingZeros[int arg0]
public static int bitCount[int arg0]
public boolean equals[java.lang.Object arg0]
public static java.lang.String toString[int arg0, int arg1]
public java.lang.String toString[]
public static java.lang.String toString[int arg0]
```

Enumerating The Fields of a Class



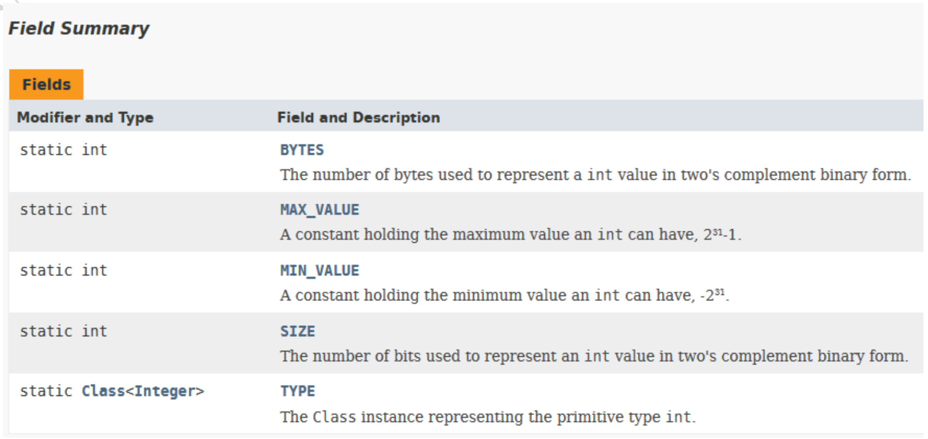
```
    private static void printFieldContents(Object AnObject) throws IllegalAccessException {

        Class c = AnObject.getClass();
        System.out.println("Contents of the fields for " + c.getName() + " class");

        for (Field f : c.getDeclaredFields()) {
            f.setAccessible(true);
            Object value = f.get(AnObject);
            System.out.println(f.getName() + ": " + value);
        }
    }
```

basics.Example3.java



Output - ReflectionSource (run)

```
run:
Contents of the fields for java.lang.Integer class
MIN_VALUE: -2147483648
MAX_VALUE: 2147483647
TYPE: int
digits: [C@52cc8049
DigitTens: [C@5b6f7412
DigitOnes: [C@27973e9b
sizeTable: [I@312b1dae
value: 10
SIZE: 32
BYTES: 4
serialVersionUID: 1360826667806852920
BUILD SUCCESSFUL (total time: 1 second)
```



Field Summary

Fields

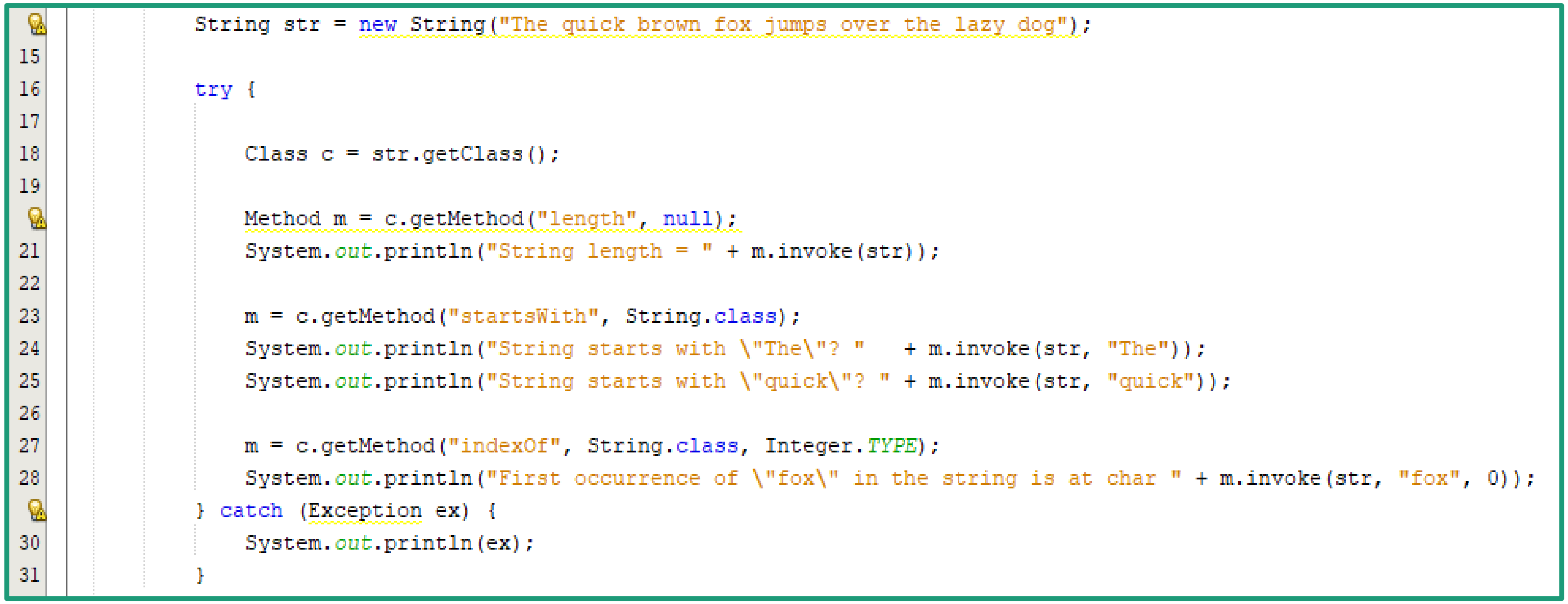
Modifier and Type	Field and Description
static int	**BYTES** The number of bytes used to represent a int value in two's complement binary form.
static int	**MAX_VALUE** A constant holding the maximum value an int can have, 2^{31}-1.
static int	**MIN_VALUE** A constant holding the minimum value an int can have, -2^{31}.
static int	**SIZE** The number of bits used to represent an int value in two's complement binary form.
static Class<Integer>	**TYPE** The Class instance representing the primitive type int.

Enumerating The Fields of a Class

- Once a field is accessible you can also set it. For example:

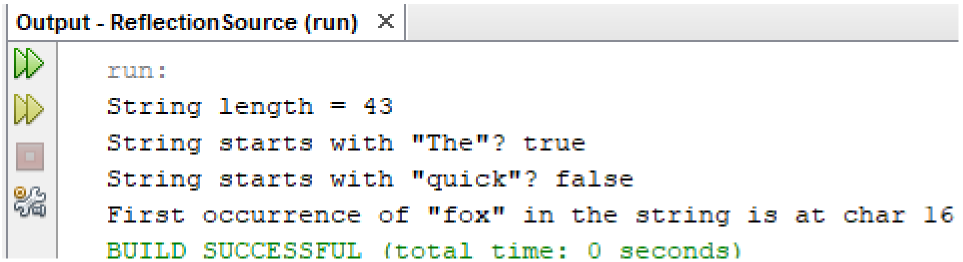
```java
Field f = c.getDeclaredField("salary");
f.setAccessible(true);
double value = f.getDouble(AnObject);
f.setDouble(AnObject, value * 1.1);
```

Invoking Methods



```
        String str = new String("The quick brown fox jumps over the lazy dog");

        try {

            Class c = str.getClass();

            Method m = c.getMethod("length", null);
            System.out.println("String length = " + m.invoke(str));

            m = c.getMethod("startsWith", String.class);
            System.out.println("String starts with \"The\"? "   + m.invoke(str, "The"));
            System.out.println("String starts with \"quick\"? " + m.invoke(str, "quick"));

            m = c.getMethod("indexOf", String.class, Integer.TYPE);
            System.out.println("First occurrence of \"fox\" in the string is at char " + m.invoke(str, "fox", 0));
        } catch (Exception ex) {
            System.out.println(ex);
        }
```

basics.Example4.java

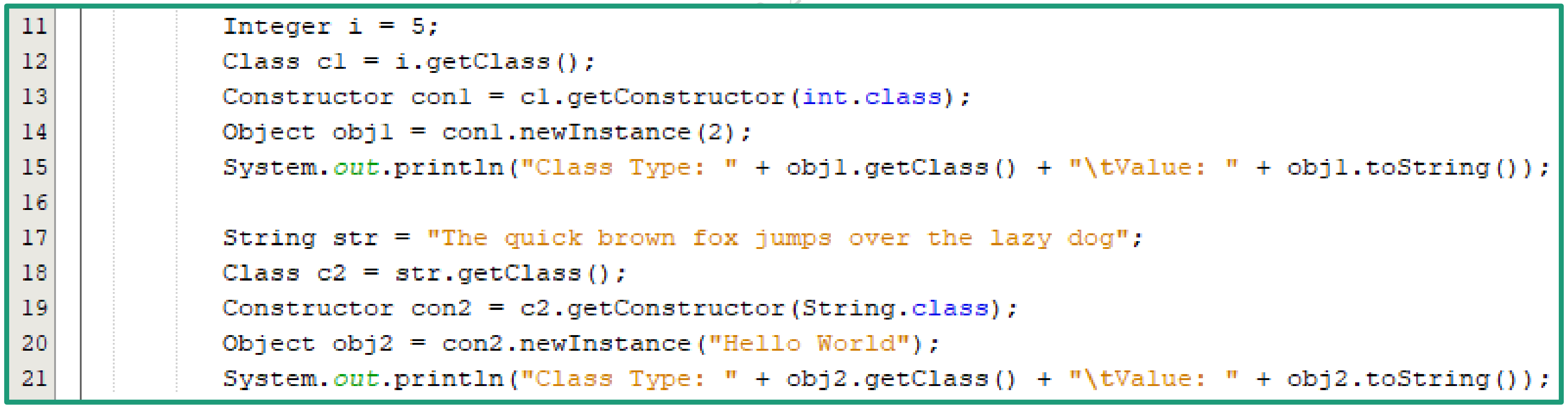


Output - ReflectionSource (run)

```
run:
String length = 43
String starts with "The"? true
String starts with "quick"? false
First occurrence of "fox" in the string is at char 16
BUILD SUCCESSFUL (total time: 0 seconds)
```

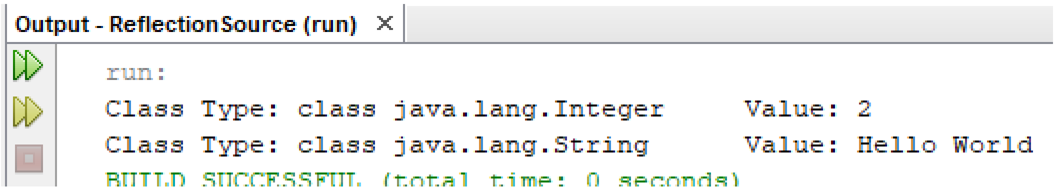
Constructing Objects

- To construct an object, first find the Constructor object and then call its newInstance method.
- For example, suppose you know that a class has a public constructor whose parameter is an int (first block of code), or a String (second block).



```java
Integer i = 5;
Class c1 = i.getClass();
Constructor con1 = c1.getConstructor(int.class);
Object obj1 = con1.newInstance(2);
System.out.println("Class Type: " + obj1.getClass() + "\tValue: " + obj1.toString());

String str = "The quick brown fox jumps over the lazy dog";
Class c2 = str.getClass();
Constructor con2 = c2.getConstructor(String.class);
Object obj2 = con2.newInstance("Hello World");
System.out.println("Class Type: " + obj2.getClass() + "\tValue: " + obj2.toString());
```

basics.Example5.java

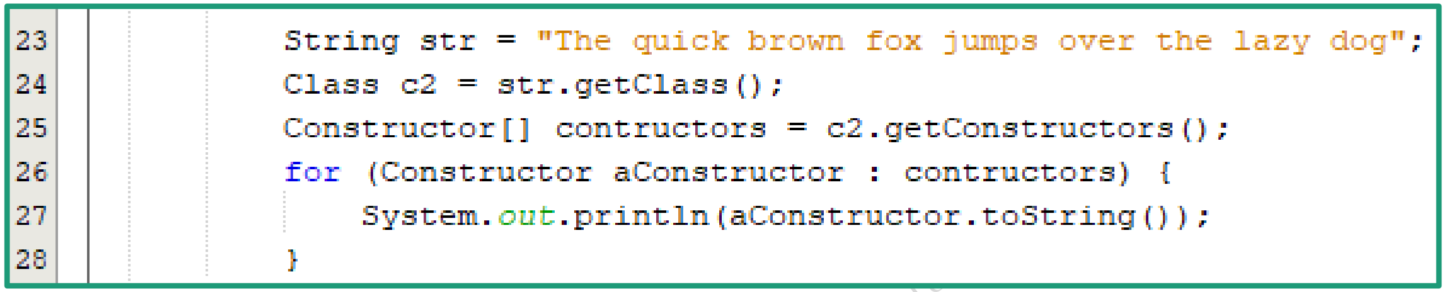


Output - ReflectionSource (run) ×

```
run:
Class Type: class java.lang.Integer     Value: 2
Class Type: class java.lang.String      Value: Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

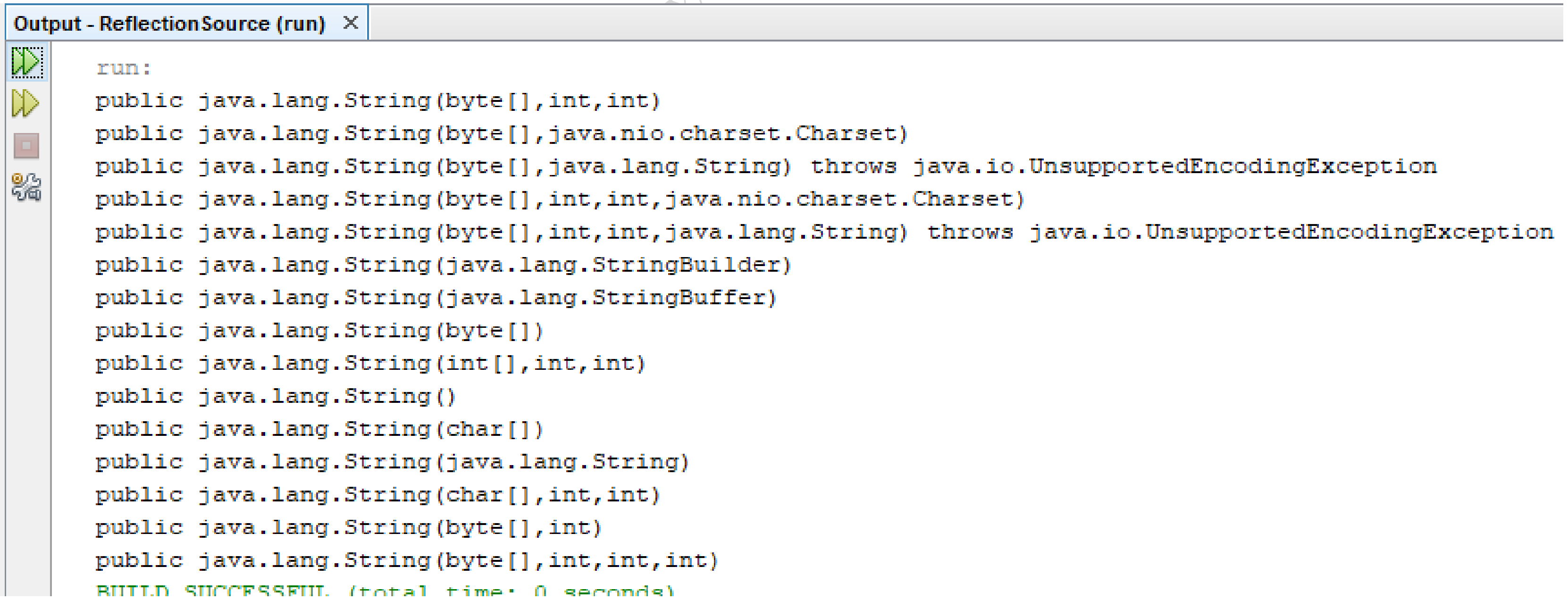
Constructing Objects

- Determining the signatures of constructors.



```
String str = "The quick brown fox jumps over the lazy dog";
Class c2 = str.getClass();
Constructor[] contructors = c2.getConstructors();
for (Constructor aConstructor : contructors) {
    System.out.println(aConstructor.toString());
}
```

basics.Example5.java



```
run:
public java.lang.String(byte[],int,int)
public java.lang.String(byte[],java.nio.charset.Charset)
public java.lang.String(byte[],java.lang.String) throws java.io.UnsupportedEncodingException
public java.lang.String(byte[],int,int,java.nio.charset.Charset)
public java.lang.String(byte[],int,int,java.lang.String) throws java.io.UnsupportedEncodingException
public java.lang.String(java.lang.StringBuilder)
public java.lang.String(java.lang.StringBuffer)
public java.lang.String(byte[])
public java.lang.String(int[],int,int)
public java.lang.String()
public java.lang.String(char[])
public java.lang.String(java.lang.String)
public java.lang.String(char[],int,int)
public java.lang.String(byte[],int)
public java.lang.String(byte[],int,int,int)
BUILD SUCCESSFUL (total time: 0 seconds)
```

Future Reading

- Completable Futures.
- Threadsafe Data Structures.
- Semaphores.

References

Y. Daniel Liang (2017) *Intro to Java Programming and Data Structures, Comprehensive Version*. 11/E. Pearson. ISBN-13 978-0134670942 (Link)

Paul J Deitel(2016) *Java How To Program*. 10/E. ISBN-13 9780134800271 (Link)

Cay S. Horstmann (2018) Core Java SE 9 For the Impatient. 2/E. ISBN-13 978-0-13-469472-6 (Link)