Java Programming
(MCA 4253)

Nested and Inner Classes

- **Nested Classes**
  - Scope of a nested class is bounded by the scope of the enclosing class
    - Example: If B is defined within A, B does not exist independently of A
  - Nested class can access all members (including private) of the class in which it is nested
  - The enclosing class cannot access members of nested class
  - Possible to declare a nested class that is local to a block

Nested and Inner Classes

- **Nested Classes** (Continued …)
  - 2 types of nested classes
    - Static:
      - A class that has *static* modifier applied
      - Can access only static members of the enclosing class
      - Cannot directly access non-static members of its enclosing class
        - It must access members of its enclosing class through an object
    - Non-static (Inner):
      - Has access to all variables and methods of its outer class
      - Can refer to them directly (similar to non-static members of the outer class)

Innerclass

```java
class Outer
{
  int outer_x = 100;

  void test()
  {
    Inner inner = new Inner();
    inner.display();
  }

  // this is an innner class
  class Inner
  {
    void display()
    {
      System.out.println("outer_x = "+outer_x);
    }
  }
}
class InnerClassDemo
{
  public static void main(String args[])
  {
    Outer outer = new Outer();
    outer.test();
  }
}
```

Nested and Inner Classes

- **Nested Classes** (Continued …)
  - Important points:
    - Instance of the inner class can be created only within the scope of the outer class
    - Compiler error if any code outside the class Outer attempts to instantiate class Inner
    - But, we can create an instance of Inner outside Outer by qualifying its name with Outer and using an Outer object as follows:

```
Outer.Inner inner = outer.new Inner();
inner.display();
```

    - An inner class has access to all members of its enclosing class, but, reverse is not true (members of inner class are known only within the scope of inner class)
    - It is possible to declare inner classes within any block scope

Inner class Demo-2:

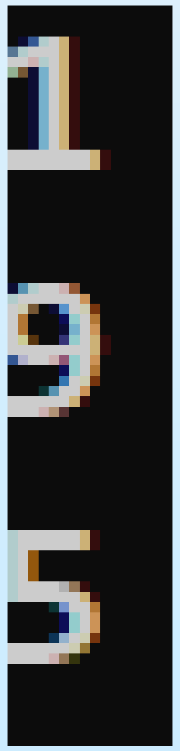
```java
class Outer
{
  int outer_x = 100;

  void test()
  {
    Inner inner = new Inner();
    inner.display();
  }

  class Inner
  {
    int y = 10; // y is local to Inner
    void display()
    {
      System.out.println("display: outer_x = " + outer_x);
    }
  }

  void showy()
  {
     System.out.println( y ); // error, y not known here!
  }
}
```

```java
class Outer
{
   int[] nums;

   Outer(int[] n)
   {
      nums = n;
   }

   void analyze()
   {
      Inner inOb = new Inner();

      System.out.println( inOb.min() );
      System.out.println( inOb.max() );
      System.out.println( inOb.avg() );
   }
```

```java
   class Inner
   {
      int min()
      {
         int m = nums[0];

         for(int i=1; i < nums.length; i++)
           if(nums[i] < m) m = nums[i];
         return m;
      }
      int max()
      {
         int m = nums[0];
         for(int i=1; i < nums.length; i++)
           if(nums[i] > m) m = nums[i];
         return m;
      }
      int avg()
      {
         int a = 0;
         for(int i=0; i < nums.length; i++)
           a += nums[i];
         return a / nums.length;
      }
   }
```

```java
class NestedClassDemo
{
  public static void main(String[] args)
  {
    int[] x = { 3, 2, 1, 5, 6, 9, 7, 8 };
    Outer outOb = new Outer(x);
    outOb.analyze();
  }
}
```

OUTPUT:



```
1
9
5
```

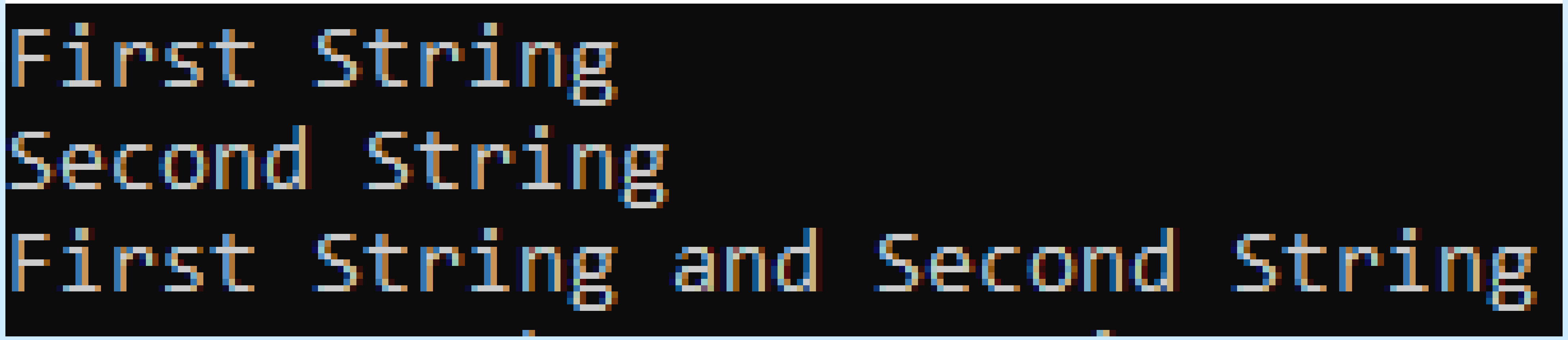
String Handling

- **String Class**
  - A *string* is an object of type **String**
  - Defined in the package java.lang as ‘final’ (cannot be inherited)
  - Not an array of characters
    - Not terminated by NULL
  - Objects are immutable (cannot be altered)
    - Fixed length
  - Two peer classes StringBuffer and StringBuilder
    - Hold strings that can be modified after they are created
  - + operator can be used for concatenating two or more strings

```
String s = “MIT” + “Manipal”;
System.out.println(“Value is ” + value);
```

String Example-1:

```java
class StringDemo
{
  public static void main(String args[])
  {
    String strOb1 = "First String";
    String strOb2 = "Second String";
    String strOb3 = strOb1 + " and " + strOb2;

    System.out.println(strOb1);
    System.out.println(strOb2);
    System.out.println(strOb3);
  }
}
```

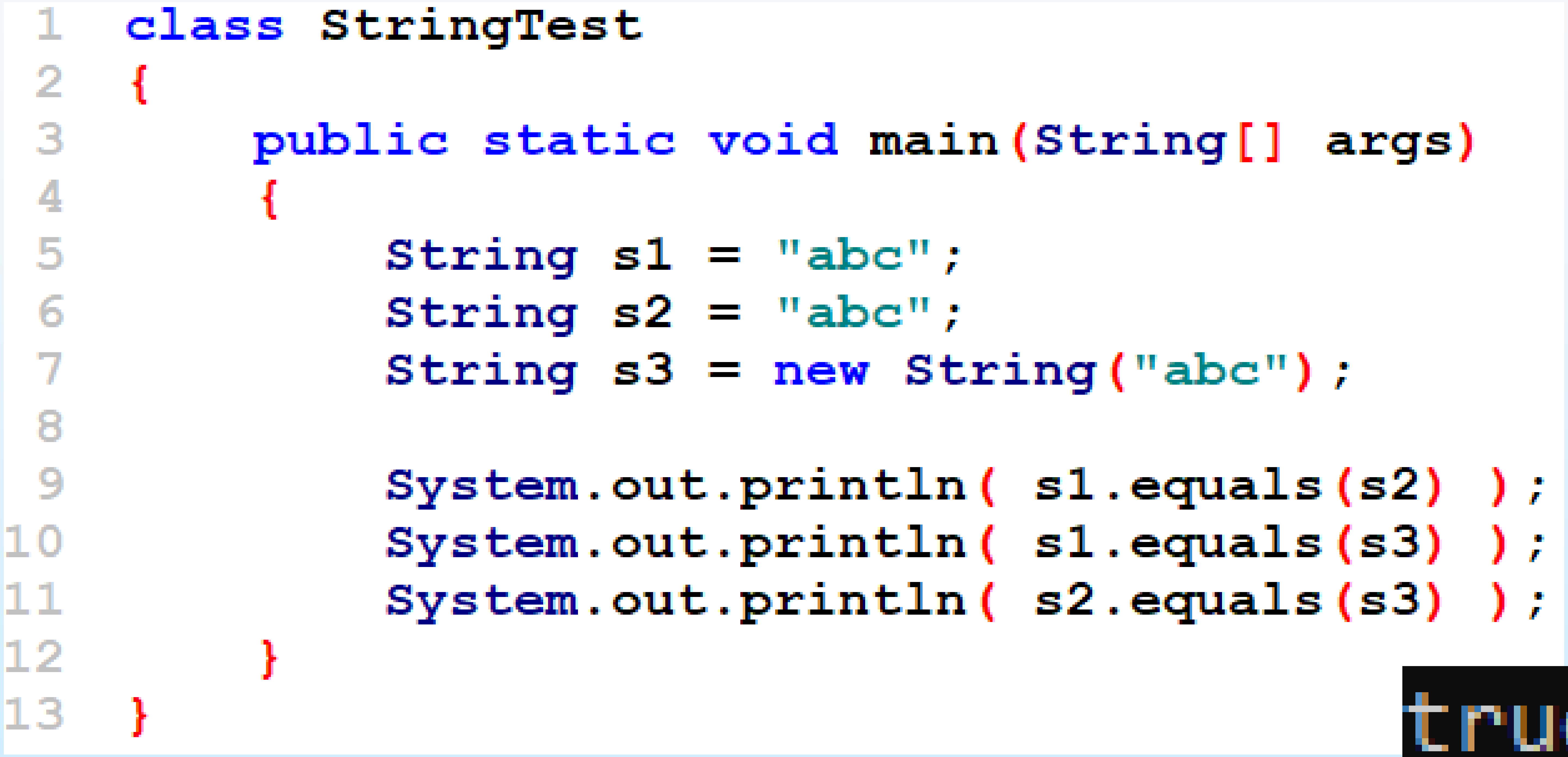


```
First String
Second String
First String and Second String
```

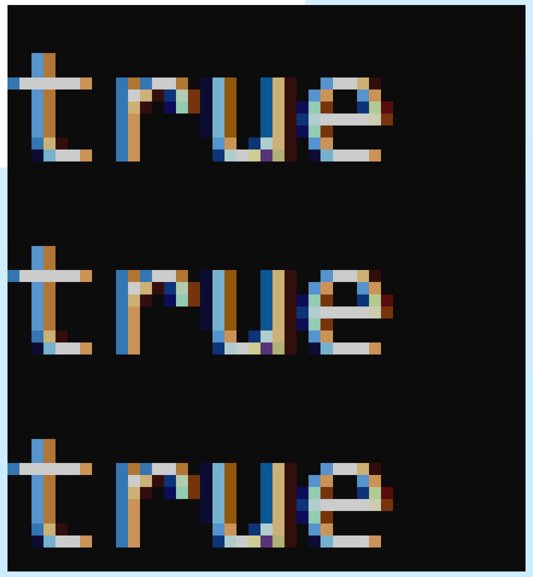
Note: The “equals()” method compares the characters within the strings.

The “==” operator compares two object references to see whether they refer to the same instance.

String Example-2:

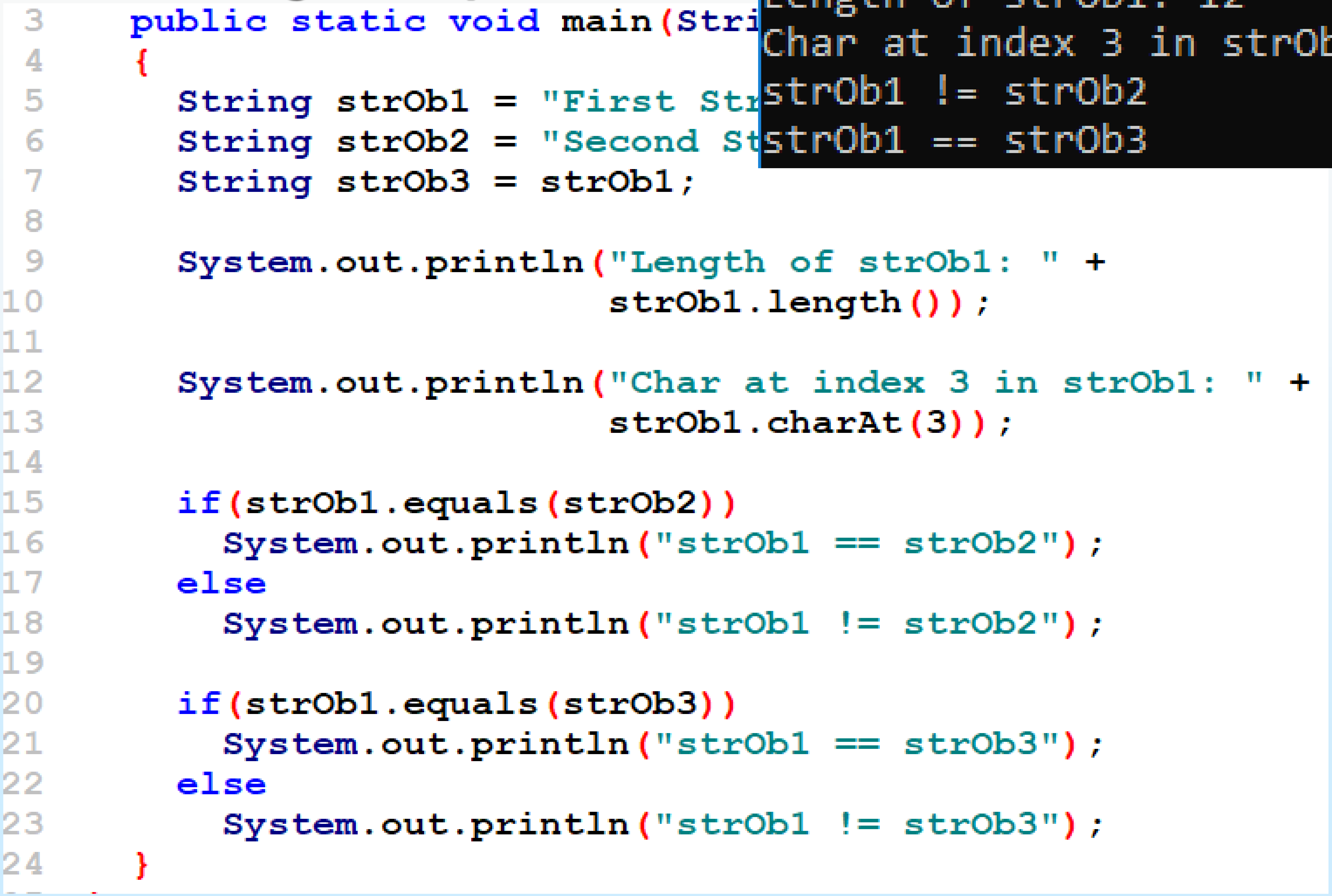


```java
class StringTest
{
      public static void main(String[] args)
      {
            String s1 = "abc";
            String s2 = "abc";
            String s3 = new String("abc");

            System.out.println( s1.equals(s2) );
            System.out.println( s1.equals(s3) );
            System.out.println( s2.equals(s3) );
      }
}
```

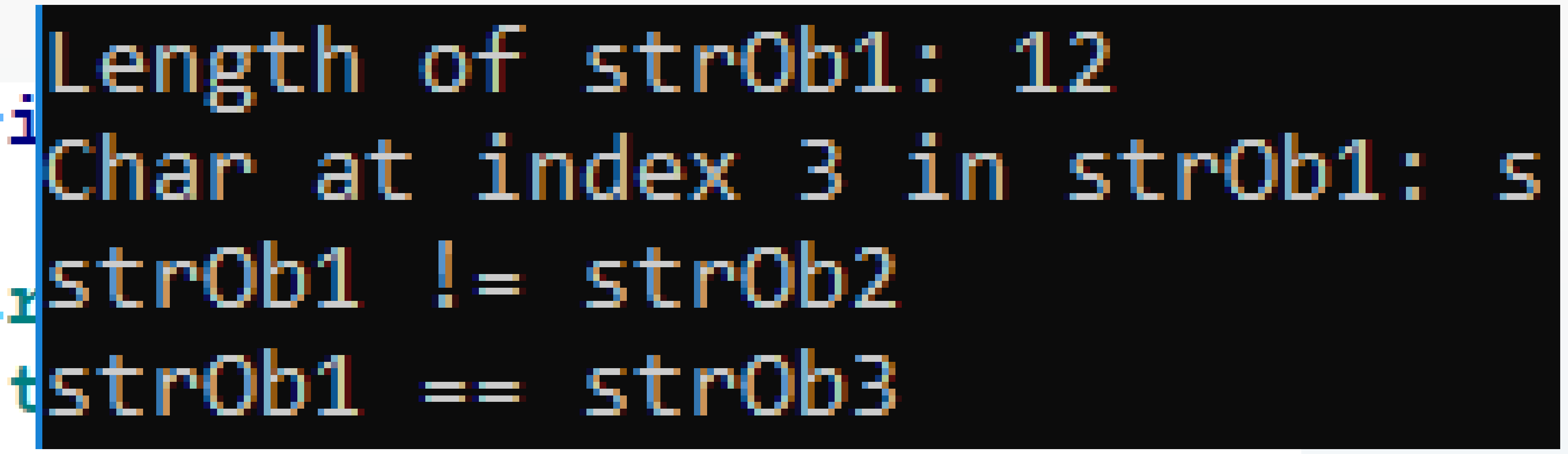


```
true
true
true
```

String Example-3:



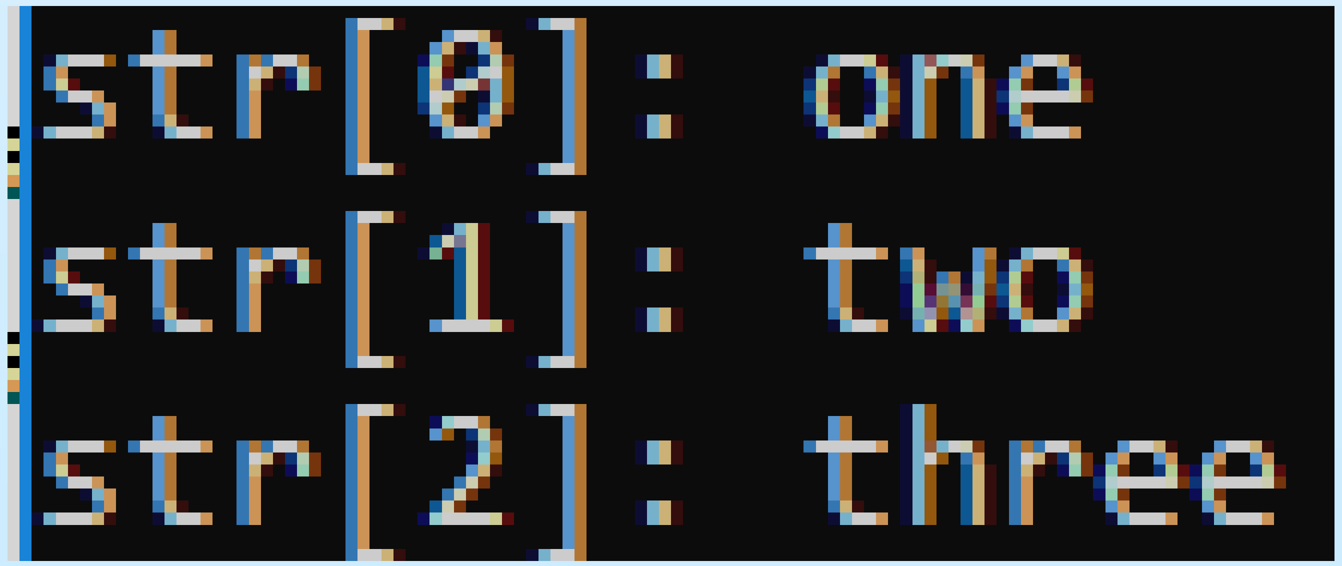
```java
    public static void main(Stri
    {
      String strOb1 = "First Str
      String strOb2 = "Second St
      String strOb3 = strOb1;

      System.out.println("Length of strOb1: " +
                                   strOb1.length());

      System.out.println("Char at index 3 in strOb1: " +
                                   strOb1.charAt(3));

      if(strOb1.equals(strOb2))
        System.out.println("strOb1 == strOb2");
      else
        System.out.println("strOb1 != strOb2");

      if(strOb1.equals(strOb3))
        System.out.println("strOb1 == strOb3");
      else
        System.out.println("strOb1 != strOb3");
    }
```



```
Length of strOb1: 12
Char at index 3 in strOb1: s
strOb1 != strOb2
strOb1 == strOb3
```

String Example-4:

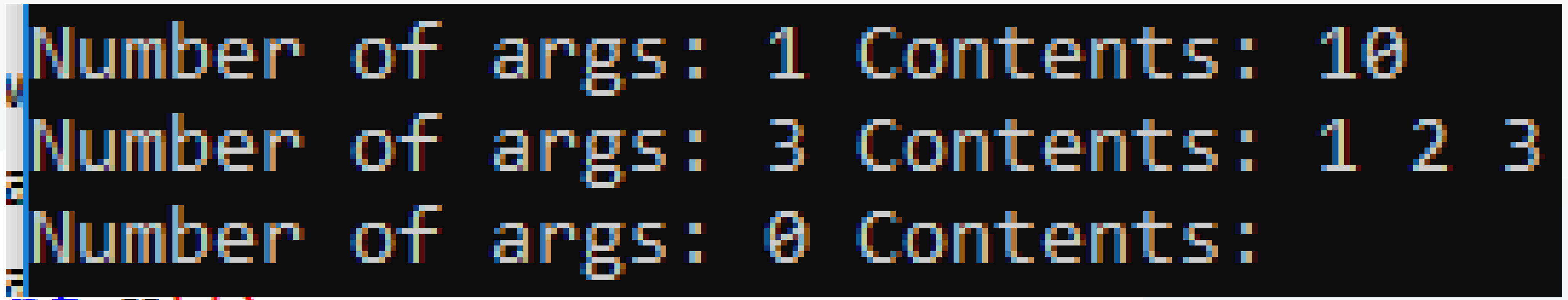
```java
class StringDemo3
{
  public static void main(String args[])
  {
    String str[] = { "one", "two", "three" };

    for(int i=0; i<str.length; i++)
      System.out.println("str[" + i + "]: " +
                                   str[i]);
  }
}
```



```
str[0]: one
str[1]: two
str[2]: three
```

String Example-5:

```java
class PassArray
{
  static void vaTest(int v[])
  {
    System.out.print("Number of args: " + v.length +
                              " Contents: ");

    for(int x : v)
      System.out.print(x + " ");

    System.out.println();
  }
public static void main(String args[])
{
    int n1[] = { 10 };
    int n2[] = { 1, 2, 3 };
    int n3[] = { };

    vaTest( n1 );
    vaTest( n2 );
    vaTest( n3 );
  }
}
```



```
Number of args: 1 Contents: 10
Number of args: 3 Contents: 1 2 3
Number of args: 0 Contents:
```

String Handling

- **String Class** (Continued …)
  - Constructors:

```
String ()
        String s1 = new String ();                  // Empty string
String (char charArray[])
String (char charArray[], int startIndex, int numChars)
        char ch[] = {'a', 'b', 'c' , 'd', 'e', 'f' };
        String s2 = new String (ch);                // "abcdef"
        String s3 = new String (ch, 2, 3);          // "cde"
```

String Handling

- **String Class** (Continued …)
  - Constructors:

```
String (String strObj)
        String s4 = new String (s3)              // “cde”
String (byte charArray[])
String (byte charArray[], int startIndex, int numChars)
        byte asc[] = { 65, 66, 67, 68, 69, 70);
        String s5 = new String (asc);            // ”ABCDEF”
        String s6 = new String (asc,2,3);        // “CDE”
```

Note: We can also create a string without using a constructor by directly assigning a value to it.

Example: String s7 = “Manipal”;

String Handling

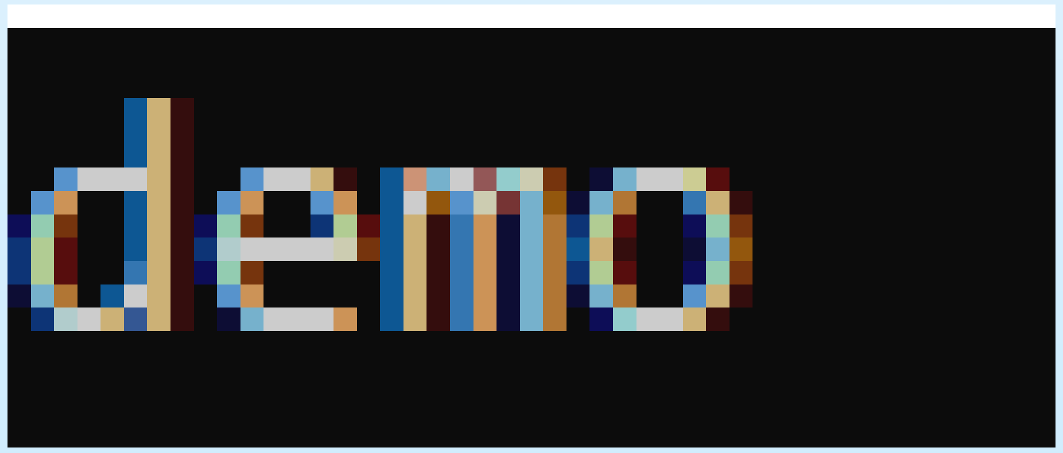
- **String Class** (Continued …)
  - String methods (Character Extraction):

char	charAt (int index)
void	getChars (int sourceStart, int sourceEnd, char target [], int targetStart)
byte [ ]	getBytes () //stores characters in an array of bytes
char [ ]	toCharArray ()

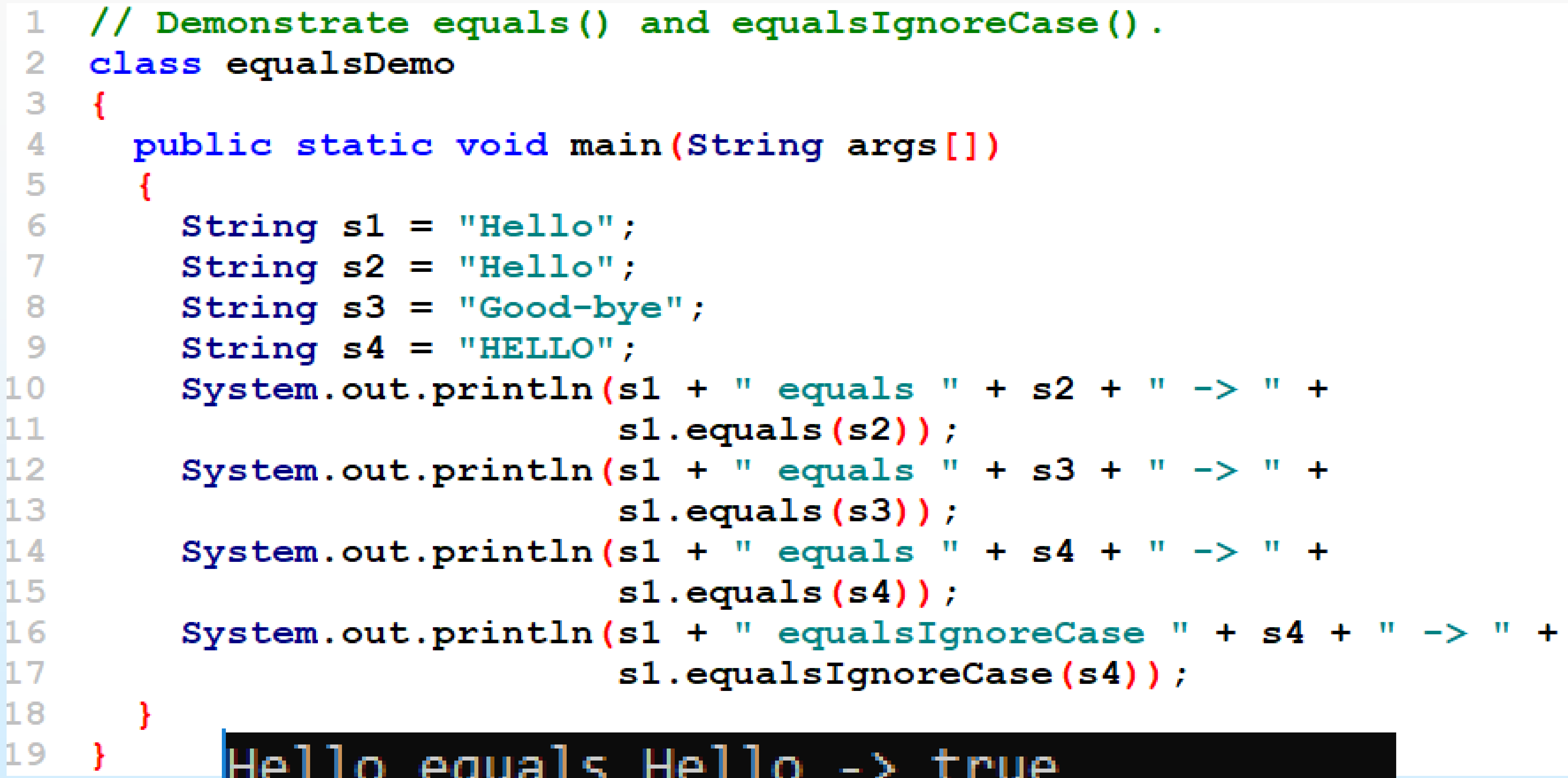
  - String methods (Case conversion):

String	toLowerCase ()
String	toUpperCase ()

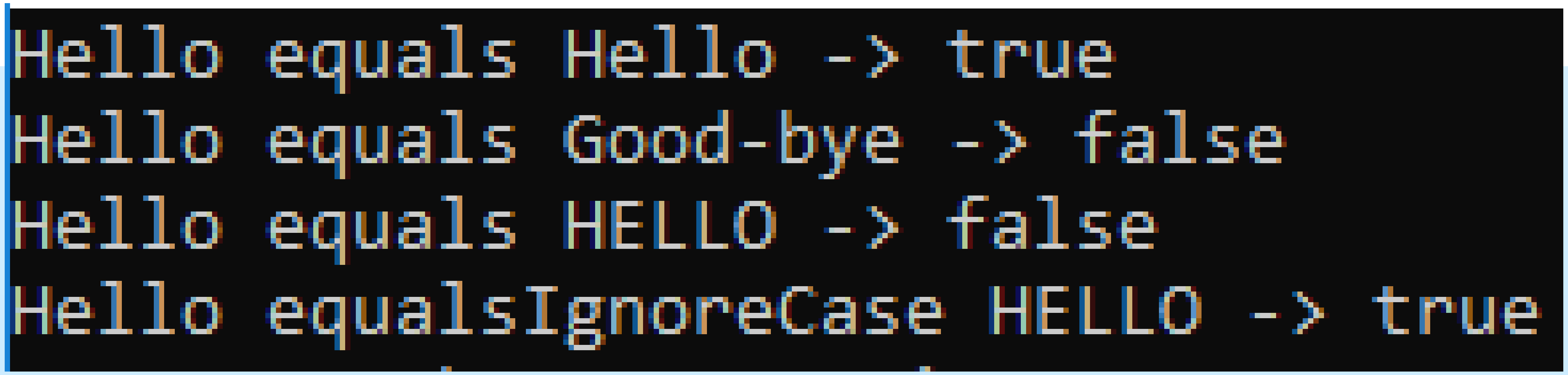
```java
class getCharsDemo
{
  public static void main(String args[])
  {
    String s = "This is a demo of the getChars method.";
    int start = 10;
    int end = 14;
    char buf[] = new char[end - start];

    s.getChars(start, end, buf, 0);
    System.out.println(buf);
  }
}
```



```
demo
```



```java
// Demonstrate equals() and equalsIgnoreCase().
class equalsDemo
{
  public static void main(String args[])
  {
    String s1 = "Hello";
    String s2 = "Hello";
    String s3 = "Good-bye";
    String s4 = "HELLO";
    System.out.println(s1 + " equals " + s2 + " -> " +
                       s1.equals(s2));
    System.out.println(s1 + " equals " + s3 + " -> " +
                       s1.equals(s3));
    System.out.println(s1 + " equals " + s4 + " -> " +
                       s1.equals(s4));
    System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> " +
                       s1.equalsIgnoreCase(s4));
  }
}
```



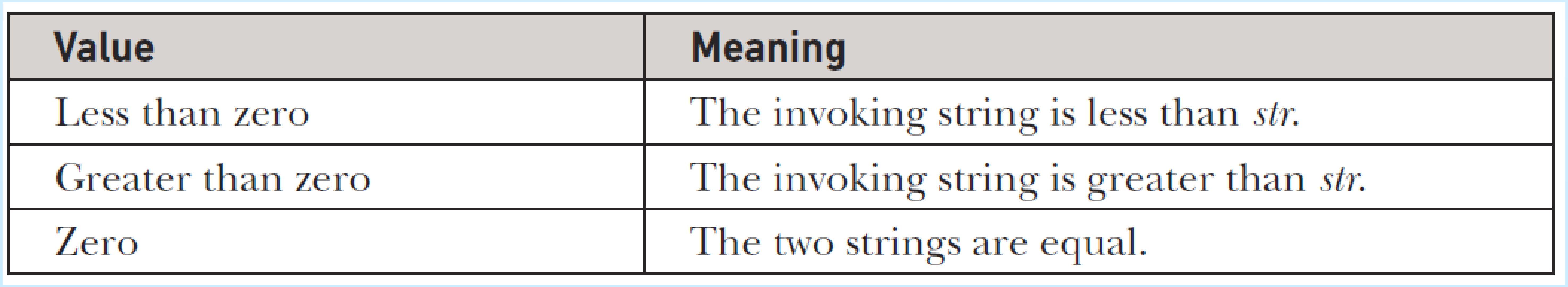
```
Hello equals Hello -> true
Hello equals Good-bye -> false
Hello equals HELLO -> false
Hello equalsIgnoreCase HELLO -> true
```

String Handling

- **String Class** (Continued …)
  - String methods (String comparison):

int	compareTo (String str)
int	compareToIgnoreCase (String str)
boolean	startsWith (String str) //determines whether a given **String** begins with a specified string.
boolean	startsWith (String str, int startIndex)
boolean	endsWith (String str)

Example: int a = S1.compareTo(str)

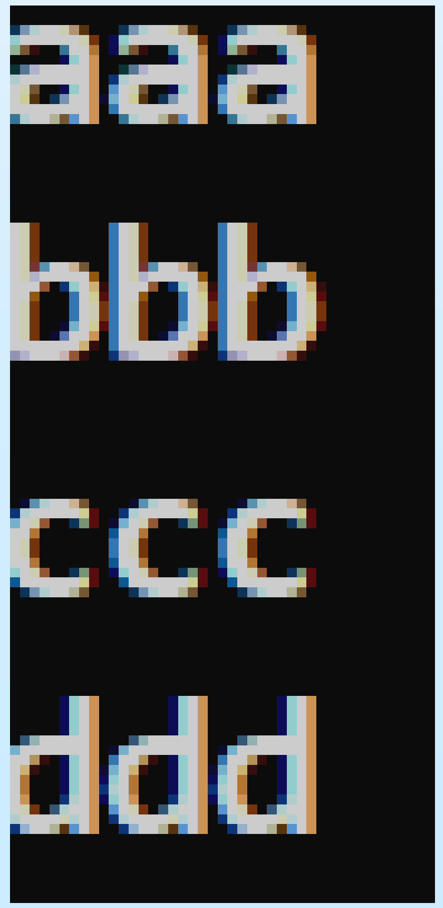


Value	Meaning
Less than zero	The invoking string is less than *str*.
Greater than zero	The invoking string is greater than *str*.
Zero	The two strings are equal.

Example: CompareTo()

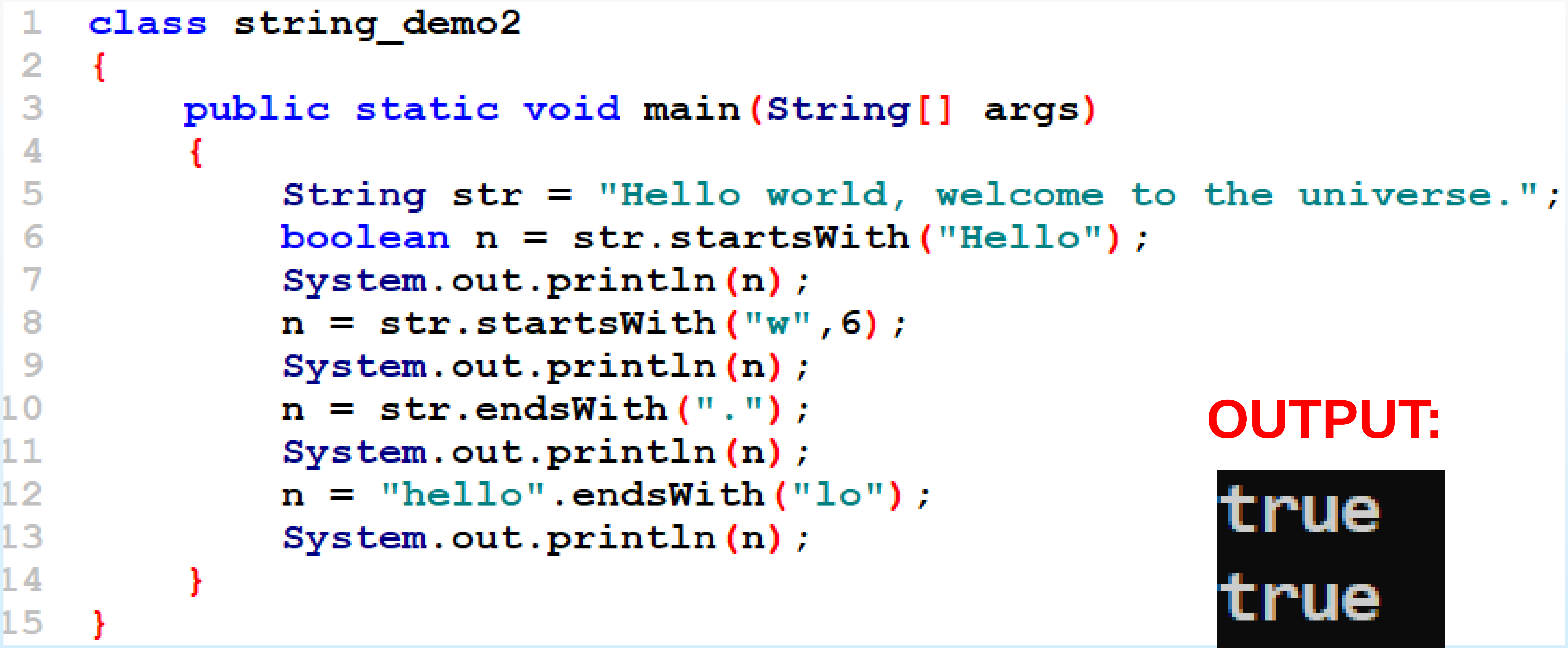
```java
class SortString
{
  public static void main(String args[])
  {
    String arr[] = { "bbb", "ddd", "ccc", "aaa" };
    for(int i = 0; i < arr.length; i++)
    {
      for(int j = i + 1; j < arr.length; j++)
      {
        if(arr[i].compareTo(arr[j]) > 0)
        {
          String t = arr[i];
          arr[i] = arr[j];
          arr[j] = t;
        }
      }
    }

    for(int i = 0; i < arr.length; i++)
        System.out.println(arr[i]);
  }
}
```

OUTPUT:



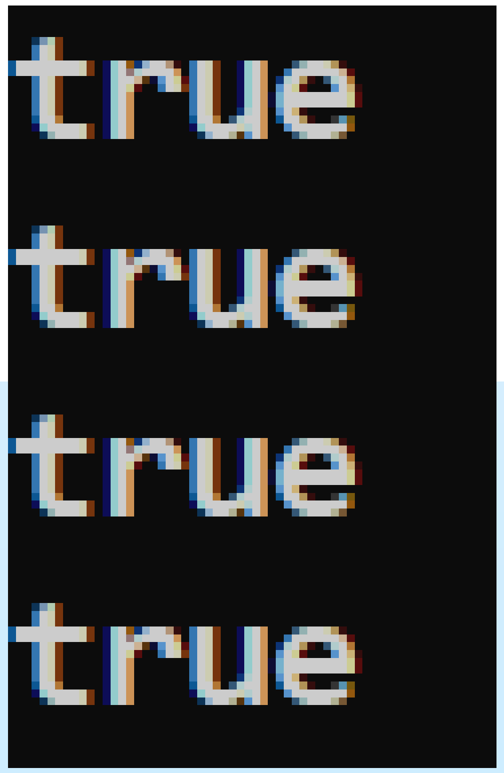
```
aaa
bbb
ccc
ddd
```

Example: startsWith() , endsWith()



```java
class string_demo2
{
    public static void main(String[] args)
    {
        String str = "Hello world, welcome to the universe.";
        boolean n = str.startsWith("Hello");
        System.out.println(n);
        n = str.startsWith("w",6);
        System.out.println(n);
        n = str.endsWith(".");
        System.out.println(n);
        n = "hello".endsWith("lo");
        System.out.println(n);
    }
}
```

OUTPUT:



```
true
true
true
true
```

String Handling

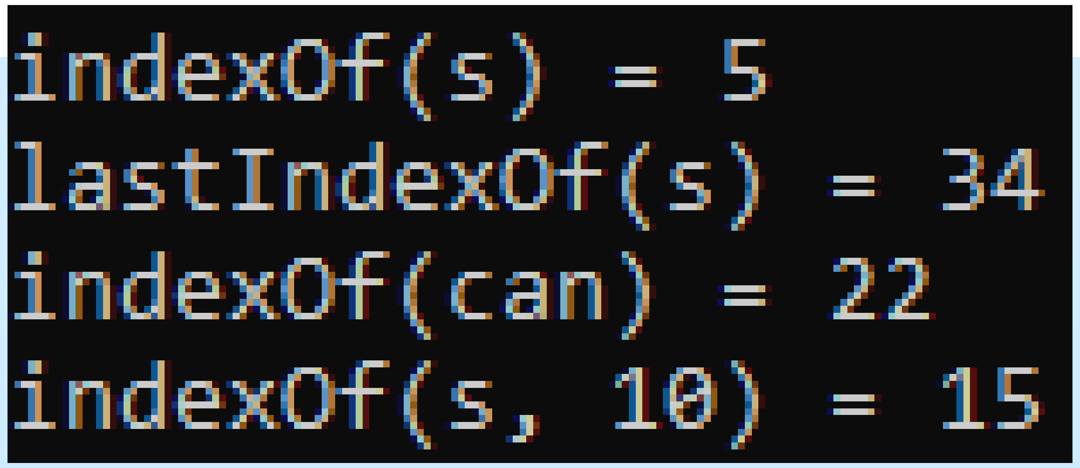
- **String Class** (Continued …)
  - String methods (Searching in a strings):

int	indexOf (char ch) //*Searches for the first occurrence of a character or substring*.
int	indexOf (char ch, int fromIndex)
int	indexOf (String str)
int	indexOf (String str, int fromIndex)
int	lastIndexOf (char ch) //*Searches for the last occurrence of a character or substring*.
int	lastIndexOf (char ch, int fromIndex)
int	lastIndexOf (String str)
int	lastIndexOf (String str, int fromIndex)
boolean	contains (CharSequence str)

Example: indexOf()

```java
class indexOfDemo
{
  public static void main(String args[])
  {
    String s = "Dont speak unless you can improve silence";

    System.out.println("\nindexOf(s) = " +
                         s.indexOf('s'));
    System.out.println("lastIndexOf(s) = " +
                         s.lastIndexOf('s'));
    System.out.println("indexOf(can) = " +
                         s.indexOf("can"));

    System.out.println("indexOf(s, 10) = " +
                         s.indexOf('s', 10));
   }
}
```

OUTPUT:

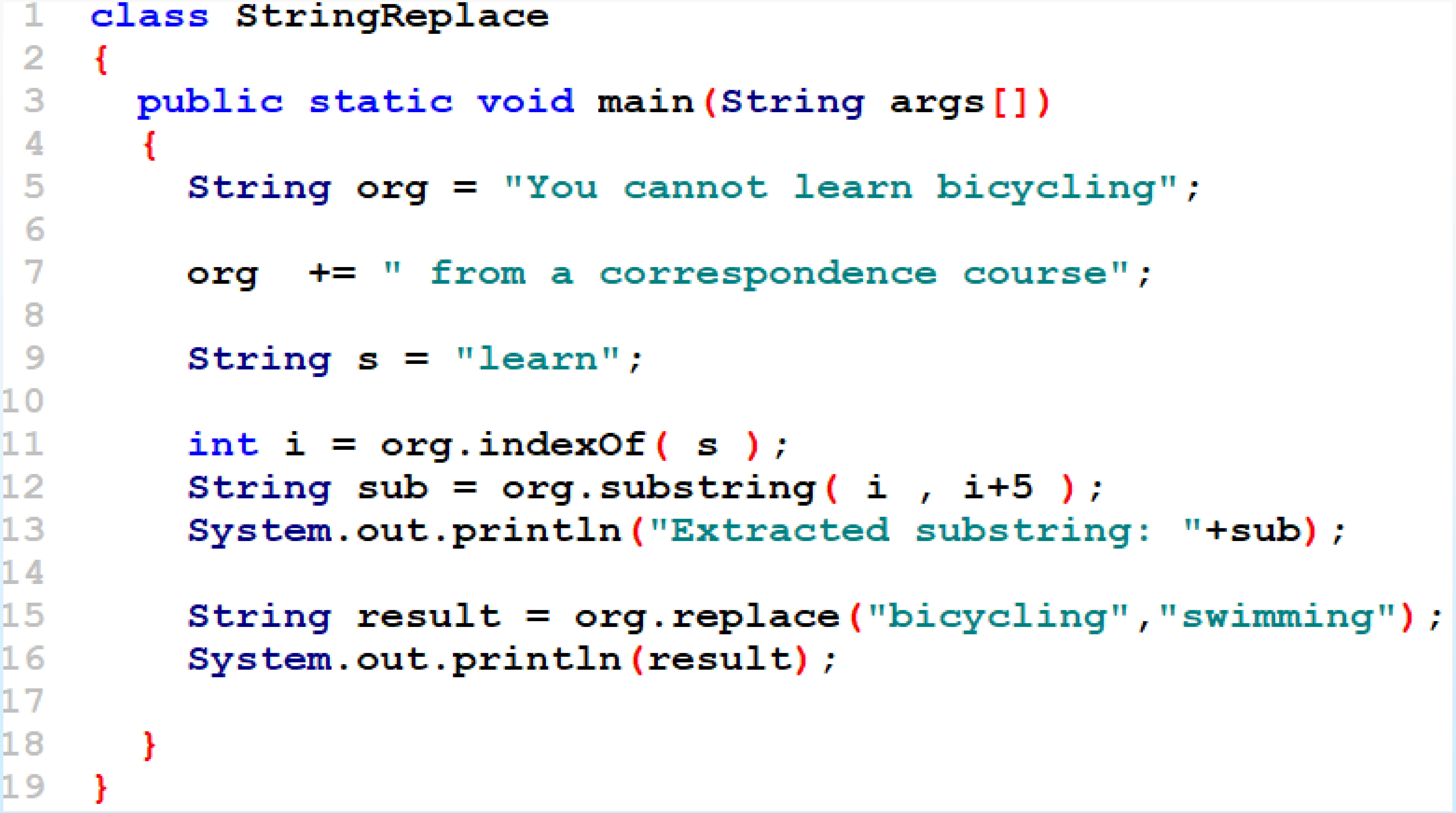


```
indexOf(s) = 5
lastIndexOf(s) = 34
indexOf(can) = 22
indexOf(s, 10) = 15
```

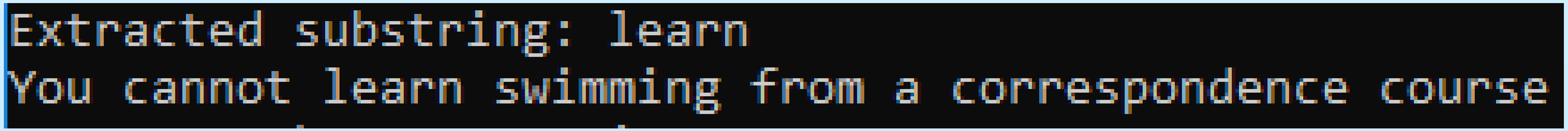
String Handling

- **String Class** (Continued …)
  - String methods (Modifying a String):

String	substring (int beginIndex) //returns a copy of the substring that begins at *startIndex*
String	substring (int beginIndex, int endIndex)
String	concat (String str)
String	replace (char oldChar, char newChar)
String	trim() //To eliminate the spaces (leading or trailing)

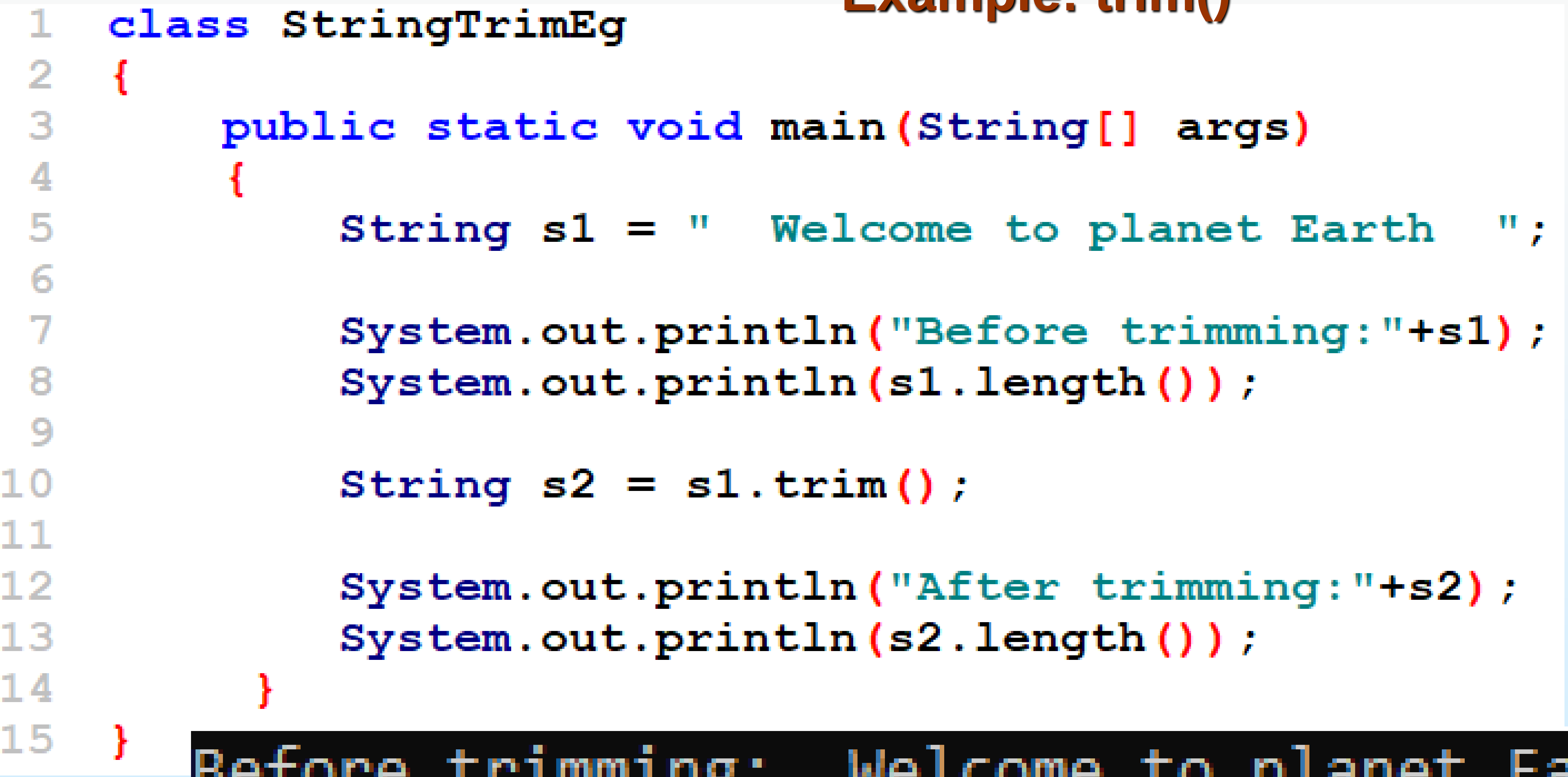


```java
class StringReplace
{
  public static void main(String args[])
  {
    String org = "You cannot learn bicycling";

    org  += " from a correspondence course";

    String s = "learn";

    int i = org.indexOf( s );
    String sub = org.substring( i , i+5 );
    System.out.println("Extracted substring: "+sub);

    String result = org.replace("bicycling","swimming");
    System.out.println(result);

  }
}
```

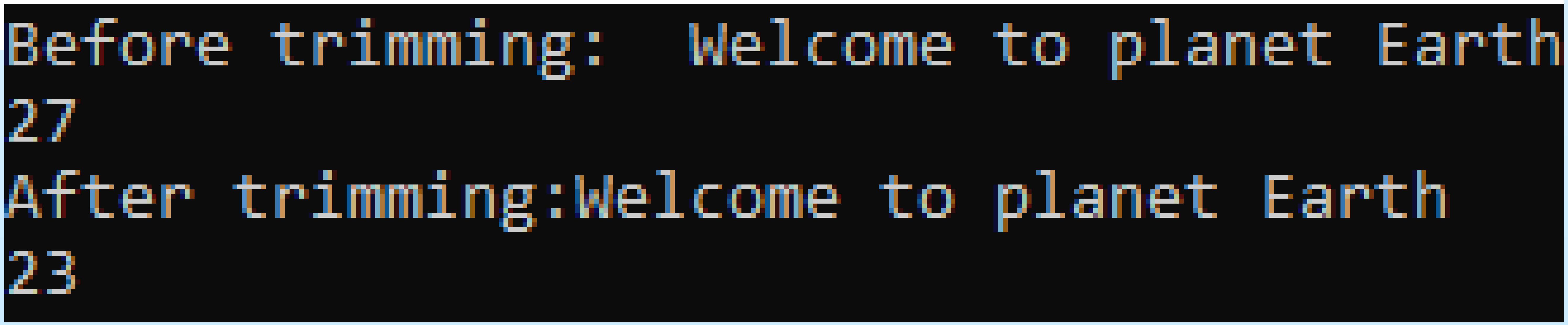


```
Extracted substring: learn
You cannot learn swimming from a correspondence course
```

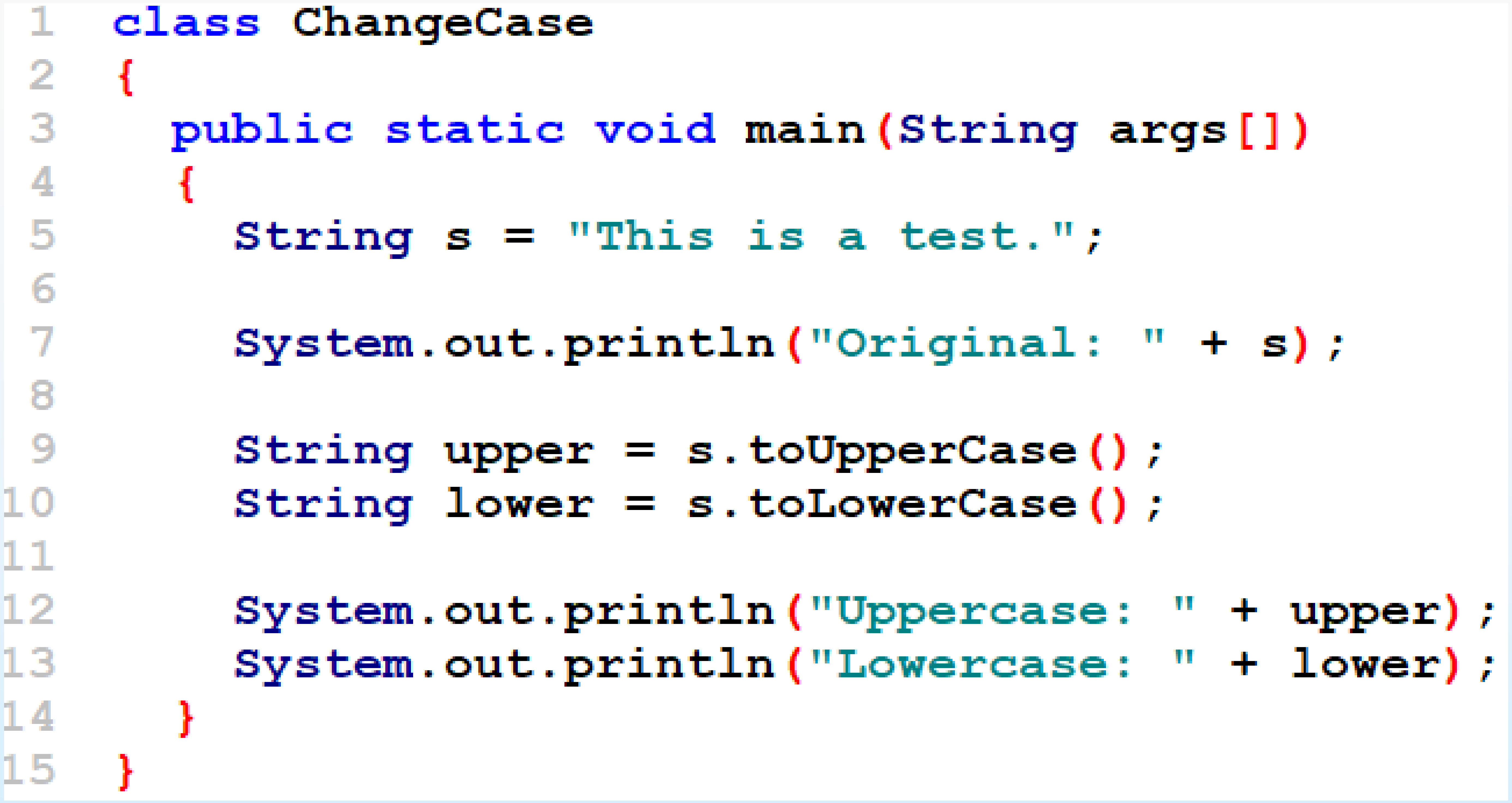
Example: trim()



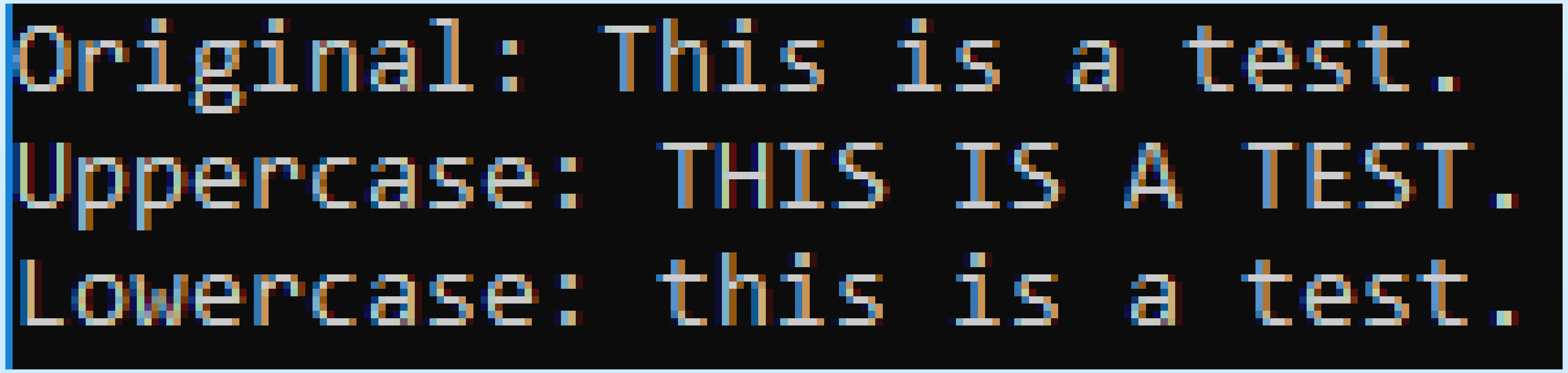
```java
class StringTrimEg
{
    public static void main(String[] args)
     {
        String s1 = "  Welcome to planet Earth  ";

        System.out.println("Before trimming:"+s1);
        System.out.println(s1.length());

        String s2 = s1.trim();

        System.out.println("After trimming:"+s2);
        System.out.println(s2.length());
      }
}
```



```
Before trimming:  Welcome to planet Earth
27
After trimming:Welcome to planet Earth
23
```



```java
class ChangeCase
{
  public static void main(String args[])
  {
    String s = "This is a test.";

    System.out.println("Original: " + s);

    String upper = s.toUpperCase();
    String lower = s.toLowerCase();

    System.out.println("Uppercase: " + upper);
    System.out.println("Lowercase: " + lower);
  }
}
```



```
Original: This is a test.
Uppercase: THIS IS A TEST.
Lowercase: this is a test.
```

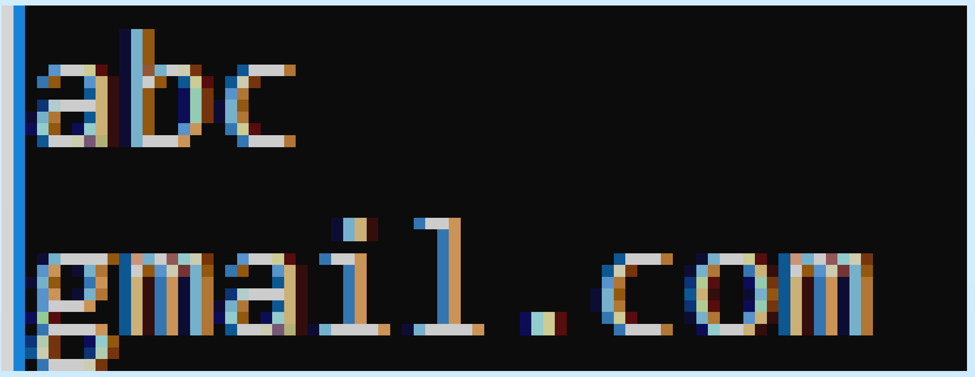
String Handling

- **String Class** (Continued …)
  - String methods (String comparison):

boolean	regionMatches (int startIndex, String str2, int str2StartIndex, int numChars)
boolean	regionMatches (boolean ignoreCase, int startIndex, String str2, int str2StartIndex, int numChars) //compares a subset of a string with a subset of another string
String [ ]	split (String regex)

Eg:

```
String str = "abc@gmail.com";
String Res[] = str.split("@");
for( String s : Res )
        System.out.println(s);
```



```
abc
gmail.com
```

```java
// Demonstrate RegionMatches.

class CompareRegions
{
  public static void main(String[] args)
  {
    String str1 = "Standing at river's edge.";
    String str2 = "Running at river's edge.";

    if(str1.regionMatches(9, str2, 8, 12))
      System.out.println("Regions match.");

    if(!str1.regionMatches(0, str2, 0, 12))
      System.out.println("Regions do not match.");
  }
}
```

OUTPUT:

```
Regions match.
Regions do not match.
```

String Handling

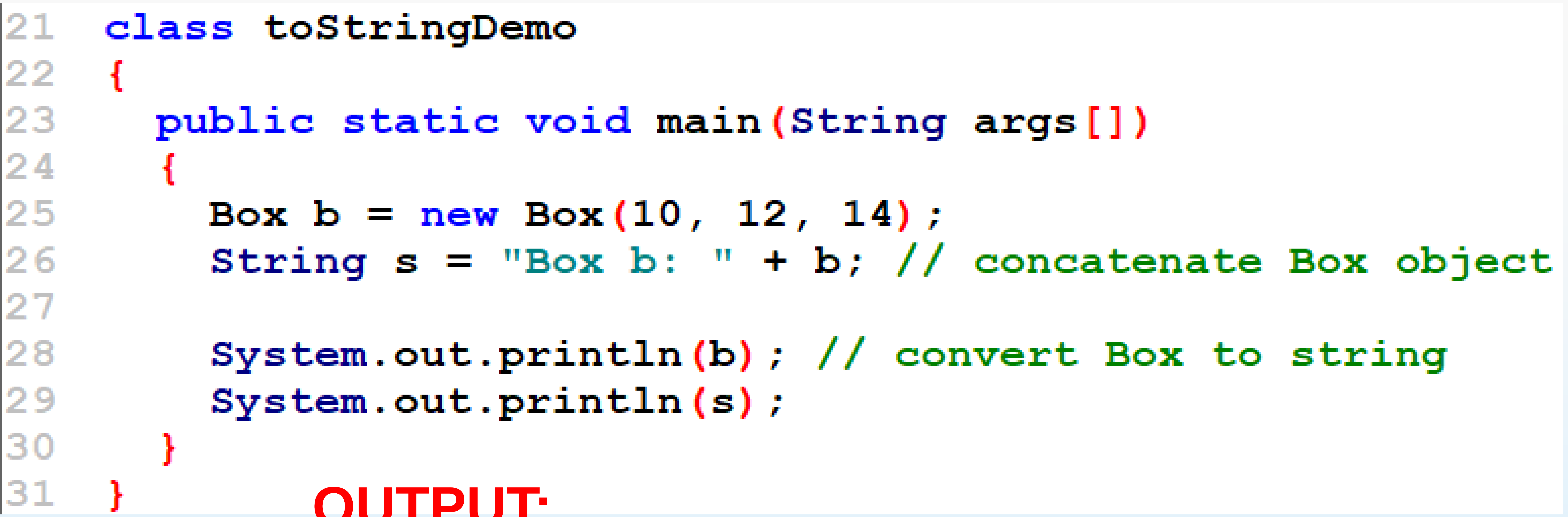
- **String Class** (Continued …)
  - String methods (Others):

int	length ()
boolean	isEmpty ()
String	toString () //
String	valueOf (double num) **// To convert the data to string format**
String	valueOf (long num)
String	valueOf (Object ob)
String	valueOf (char chars[ ], int startIndex, int num)

Note: The "valueOf()" method is a static method.

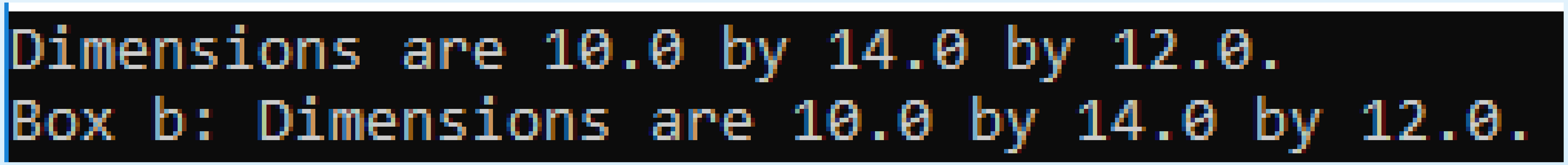
Example: ToString()

```
class Box
{
  double width;
  double height;
  double depth;

  Box(double w, double h, double d)
  {
    width = w;
    height = h;
    depth = d;
  }

  public String toString()
  {
    return "Dimensions are " + width + " by " +
            depth + " by " + height + ".";
  }
}
```



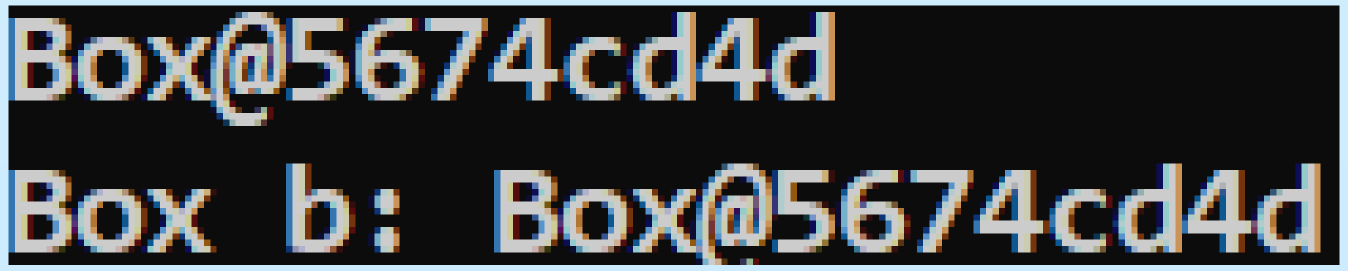
```java
class toStringDemo
{
  public static void main(String args[])
  {
    Box b = new Box(10, 12, 14);
    String s = "Box b: " + b; // concatenate Box object

    System.out.println(b); // convert Box to string
    System.out.println(s);
  }
}
```

OUTPUT:



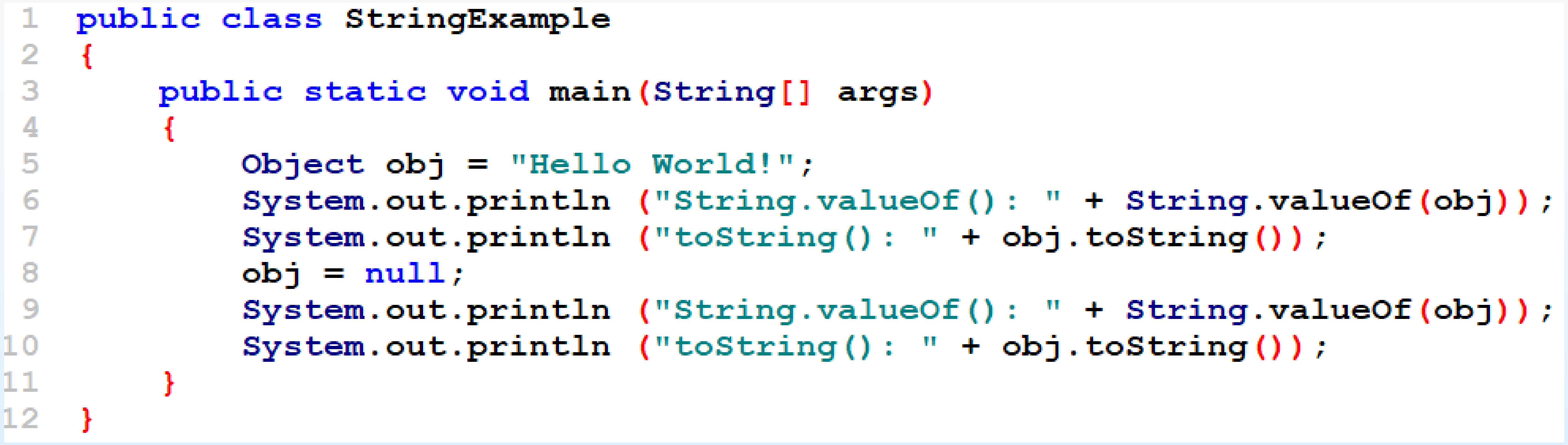
```
Dimensions are 10.0 by 14.0 by 12.0.
Box b: Dimensions are 10.0 by 14.0 by 12.0.
```

OUTPUT(without overriding toString()):



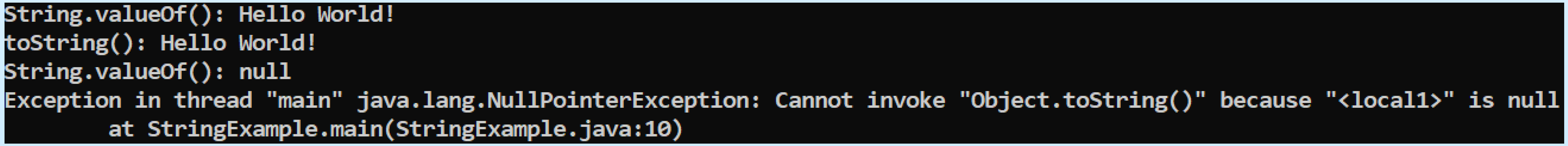
```
Box@5674cd4d
Box b: Box@5674cd4d
```

toString vs valueOf()



```
 1  public class StringExample
 2  {
 3      public static void main(String[] args)
 4      {
 5          Object obj = "Hello World!";
 6          System.out.println ("String.valueOf(): " + String.valueOf(obj));
 7          System.out.println ("toString(): " + obj.toString());
 8          obj = null;
 9          System.out.println ("String.valueOf(): " + String.valueOf(obj));
10          System.out.println ("toString(): " + obj.toString());
11      }
12  }
```

OUTPUT:



```
String.valueOf(): Hello World!
toString(): Hello World!
String.valueOf(): null
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "Object.toString()" because "<local1>" is null
        at StringExample.main(StringExample.java:10)
```

String Handling

- **String Class** (Continued …)
  - Creating format string:
    - Use String’s static format() method

  Example:

```
String fs;
int intVar = 23;
float floatVar = 5.25f;
String stringVar = “Computer Applications”;
fs = String.format (”Values are %d %f %s“, intVar, floatVar, stringVar);
System.out.println (fs);
```

Output:

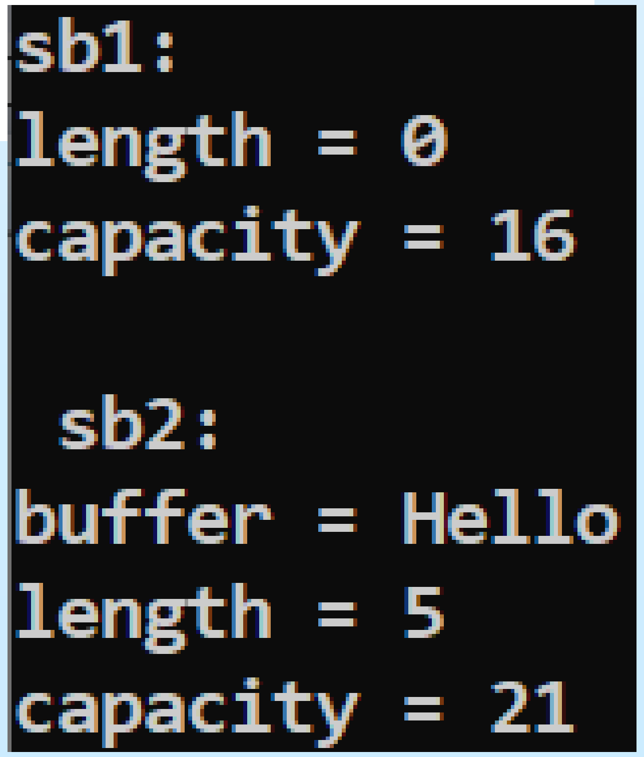
```
Values are 23 5.25 Computer Applications
```

String Handling

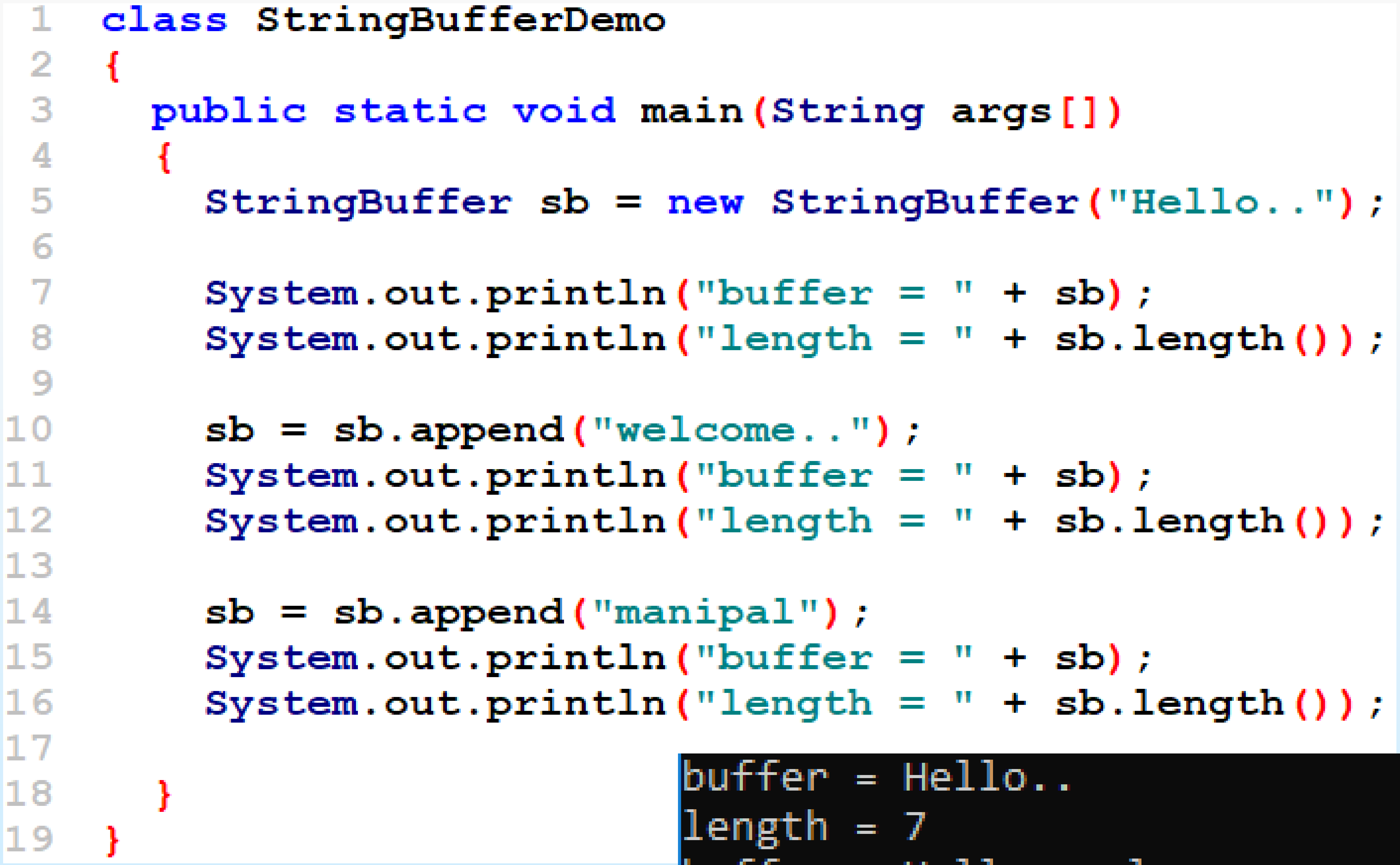
- **StringBuffer Class**
  - Strings that can be altered both in terms of length and content
  - StringBuffer constructors:

StringBuffer ()	// creates an empty string buffer with the initial capacity of 16.
StringBuffer (int capacity)	// creates an empty string buffer with the specified capacity as length.
StringBuffer (String str)	//creates a string buffer with the specified string.
StringBuffer (CharSequence chars)	

```java
// StringBuffer length vs. capacity.
class StringBufferDemo1
{
    public static void main(String args[])
    {
        StringBuffer sb1 = new StringBuffer();
        StringBuffer sb2 = new StringBuffer("Hello");

        System.out.println("sb1:");
        System.out.println("length = " + sb1.length());
        System.out.println("capacity = " + sb1.capacity());

        System.out.println("\n sb2:");
        System.out.println("buffer = " +sb2);
        System.out.println("length = " +sb2.length());
        System.out.println("capacity = " +sb2.capacity());
    }
}
```

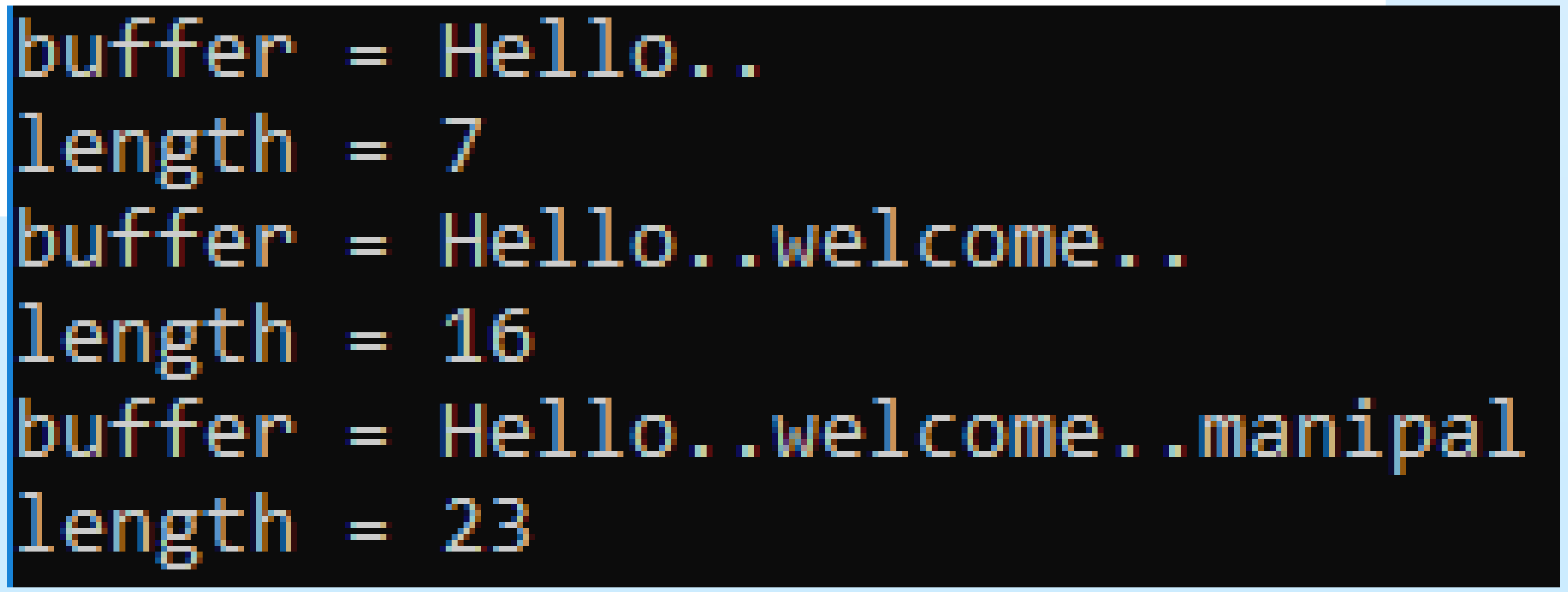


```
sb1:
length = 0
capacity = 16

 sb2:
buffer = Hello
length = 5
capacity = 21
```



```java
class StringBufferDemo
{
  public static void main(String args[])
  {
    StringBuffer sb = new StringBuffer("Hello..");

    System.out.println("buffer = " + sb);
    System.out.println("length = " + sb.length());

    sb = sb.append("welcome..");
    System.out.println("buffer = " + sb);
    System.out.println("length = " + sb.length());

    sb = sb.append("manipal");
    System.out.println("buffer = " + sb);
    System.out.println("length = " + sb.length());

  }
}
```

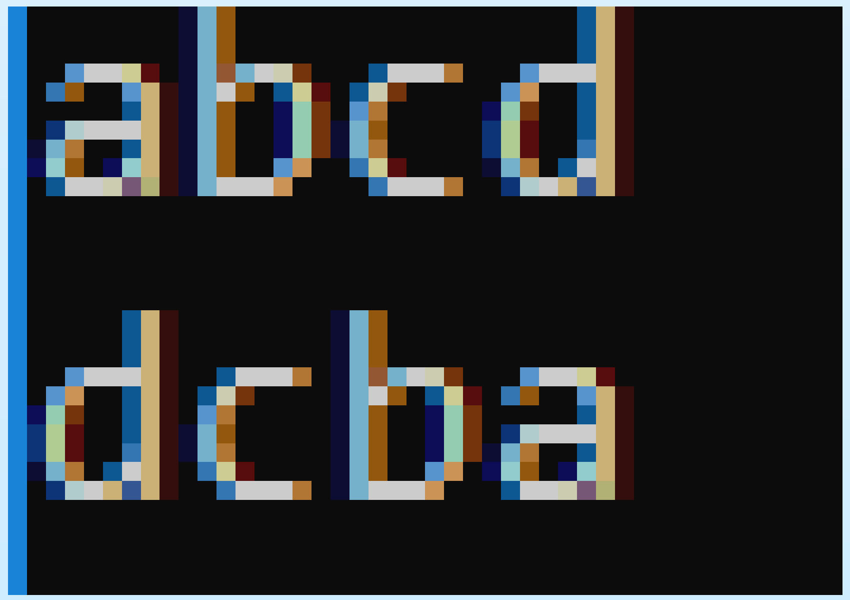
OUTPUT:



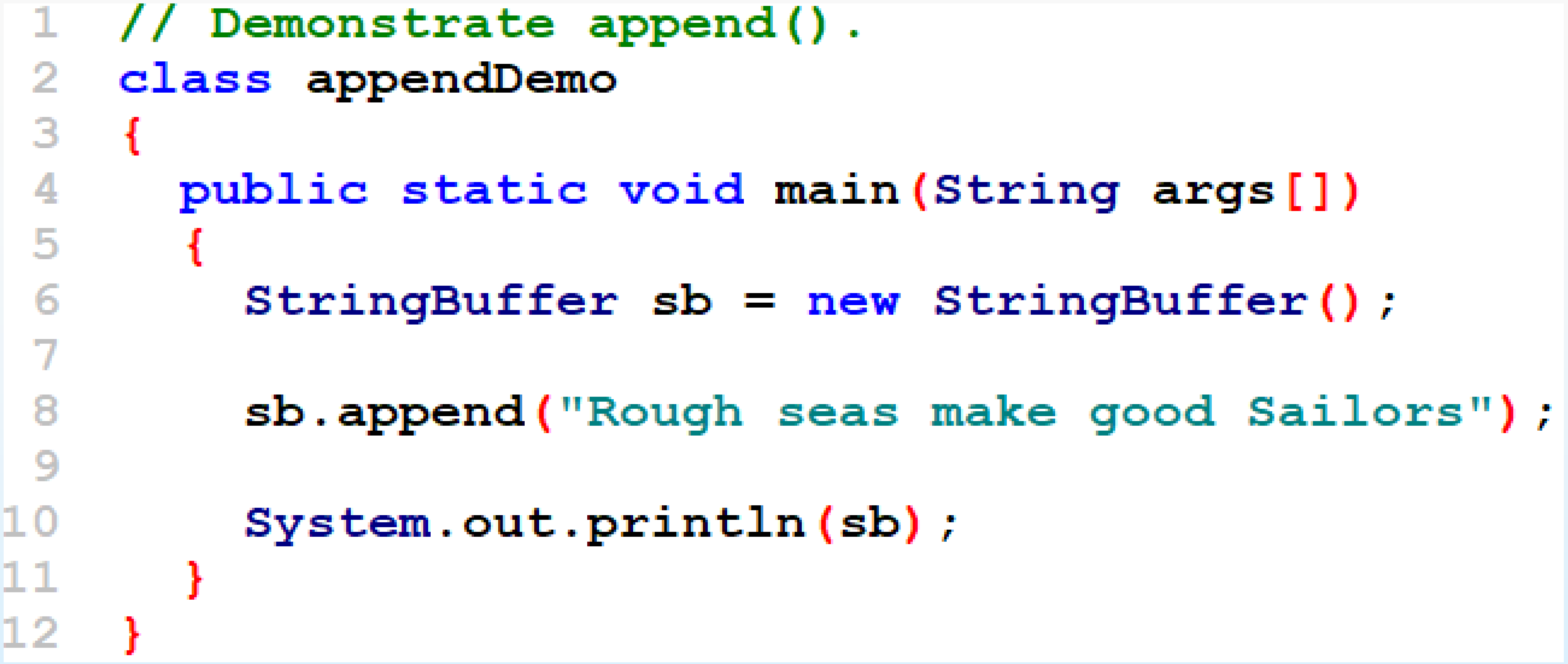
```
buffer = Hello..
length = 7
buffer = Hello..welcome..
length = 16
buffer = Hello..welcome..manipal
length = 23
```

```java
// Using reverse() to reverse a StringBuffer.
class ReverseDemo
{
  public static void main(String args[])
  {
    StringBuffer s = new StringBuffer("abcd");

    System.out.println(s);
    s.reverse();
    System.out.println(s);
  }
}
```

OUTPUT:

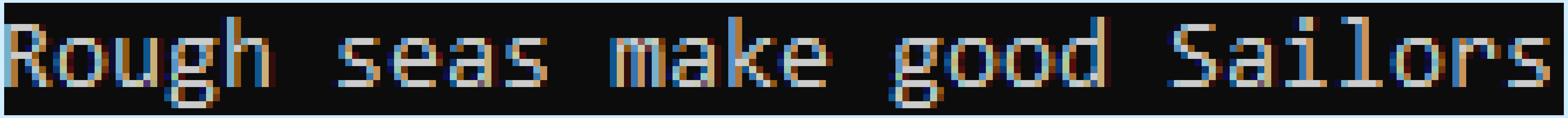


```
abcd
dcba
```



```java
// Demonstrate append().
class appendDemo
{
  public static void main(String args[])
  {
    StringBuffer sb = new StringBuffer();

    sb.append("Rough seas make good Sailors");

    System.out.println(sb);
  }
}
```

OUTPUT:



```
Rough seas make good Sailors
```

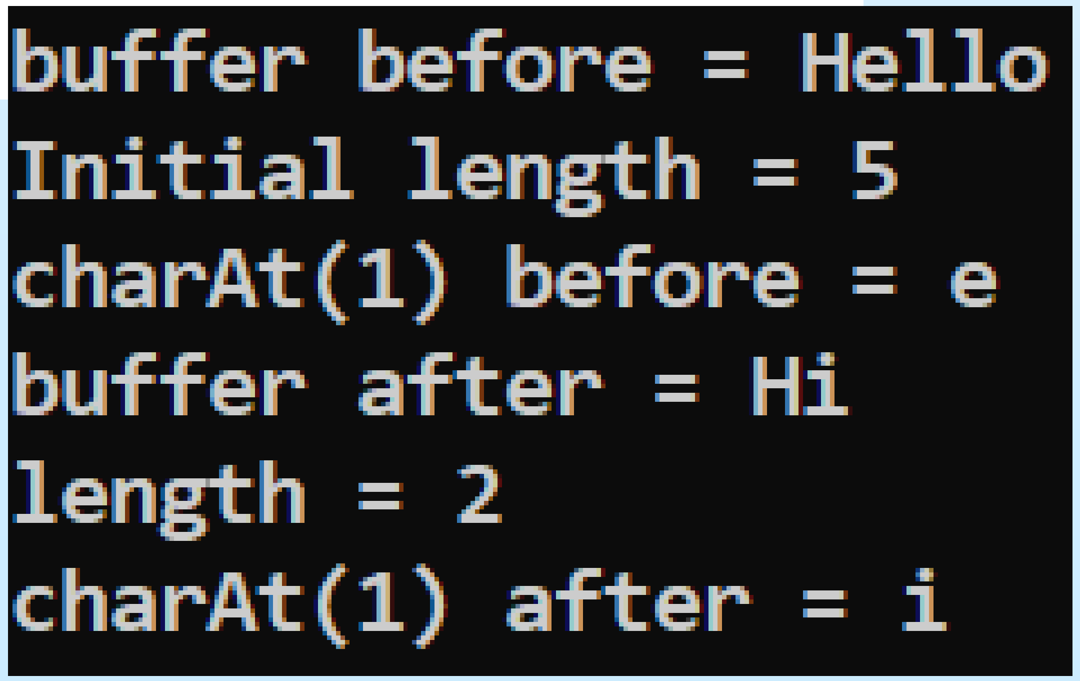
String Handling

- **StringBuffer Class** (Continued ...)
  - StringBuffer Methods:

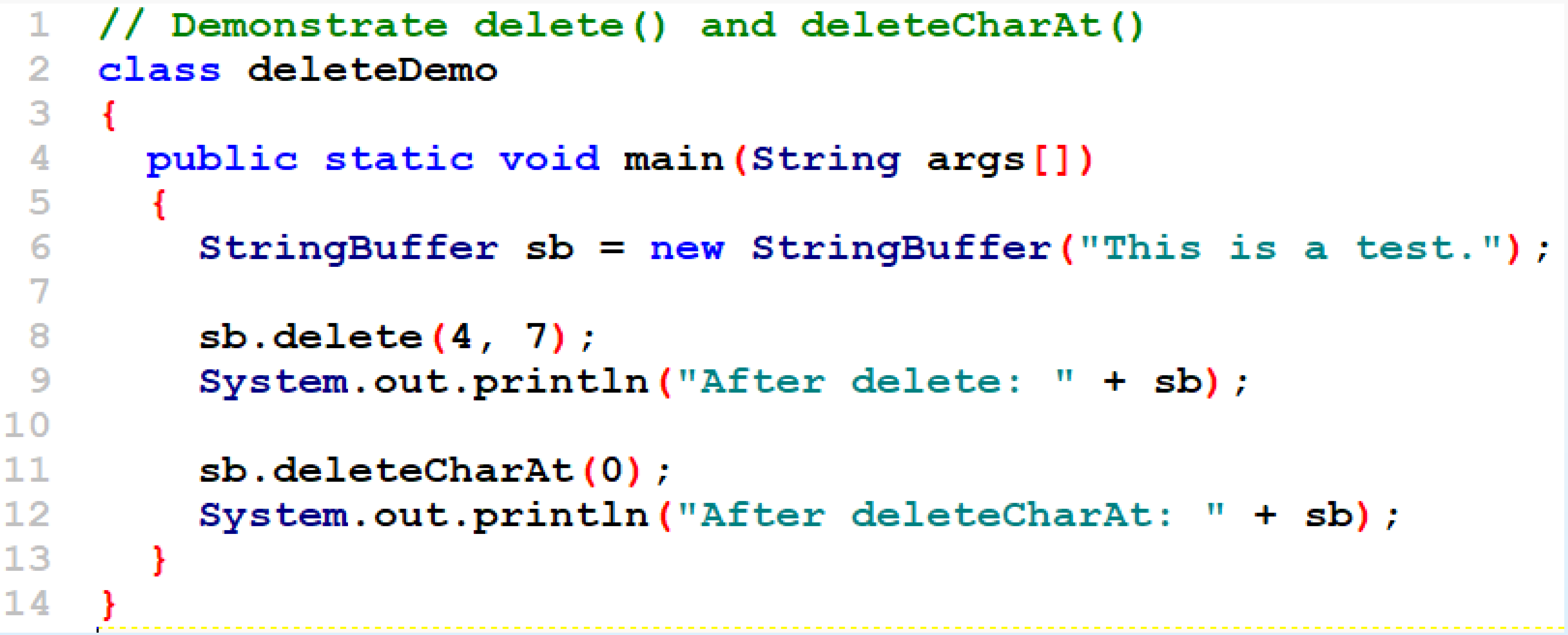
void	setCharAt (int index, char ch) // **to set the specified character at the given index**
void	setLength (int newLength) //**To set the length of the string within a StringBuffer object**
String	substring (int start)
String	substring (int start, int end)
void	trimToSize()
StringBuffer	delete (int start, int end)
StringBuffer	deleteCharAt (int index)

```java
// Demonstrate charAt() and setCharAt().
class setCharAtDemo
{
  public static void main(String args[])
  {
    StringBuffer sb = new StringBuffer("Hello");
    System.out.println("buffer before = " + sb);
    System.out.println("Initial length = " +sb.length() );

    System.out.println("charAt(1) before = " + sb.charAt(1));

    sb.setCharAt(1, 'i');
    sb.setLength(2);

    System.out.println("buffer after = " + sb);
    System.out.println("length = " +sb.length() );
    System.out.println("charAt(1) after = " + sb.charAt(1));
  }
}
```

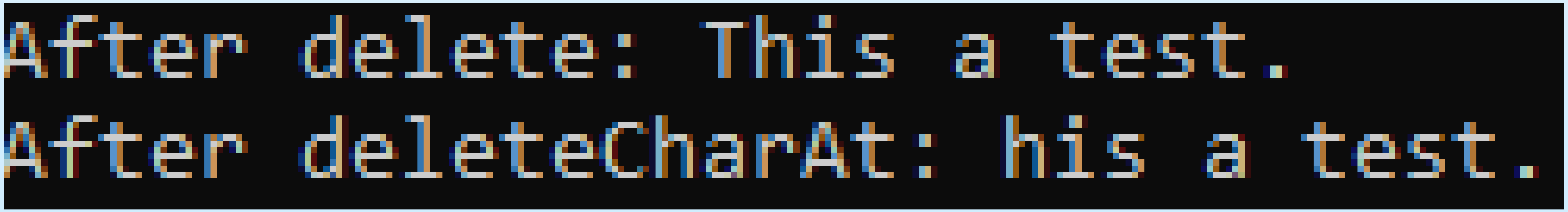
OUTPUT:



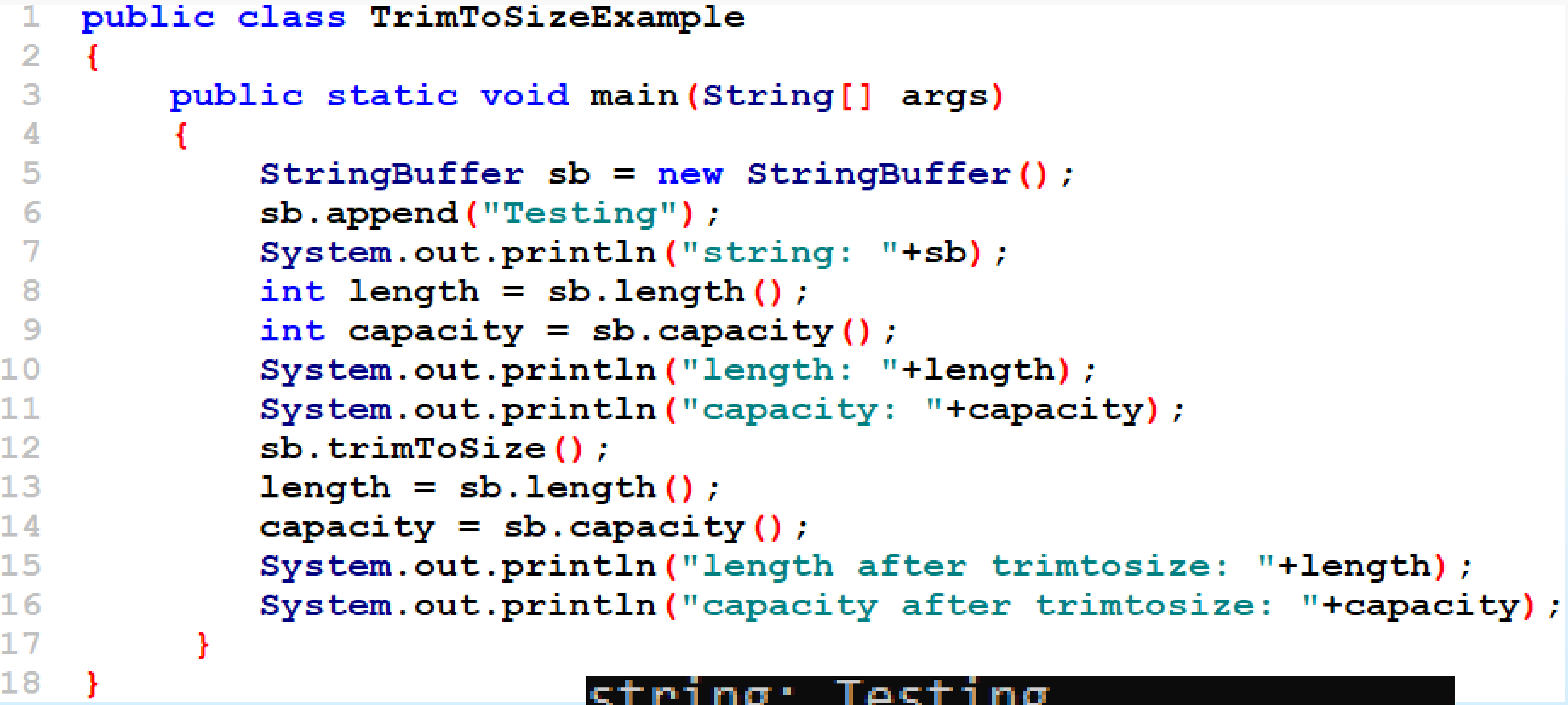
```
buffer before = Hello
Initial length = 5
charAt(1) before = e
buffer after = Hi
length = 2
charAt(1) after = i
```



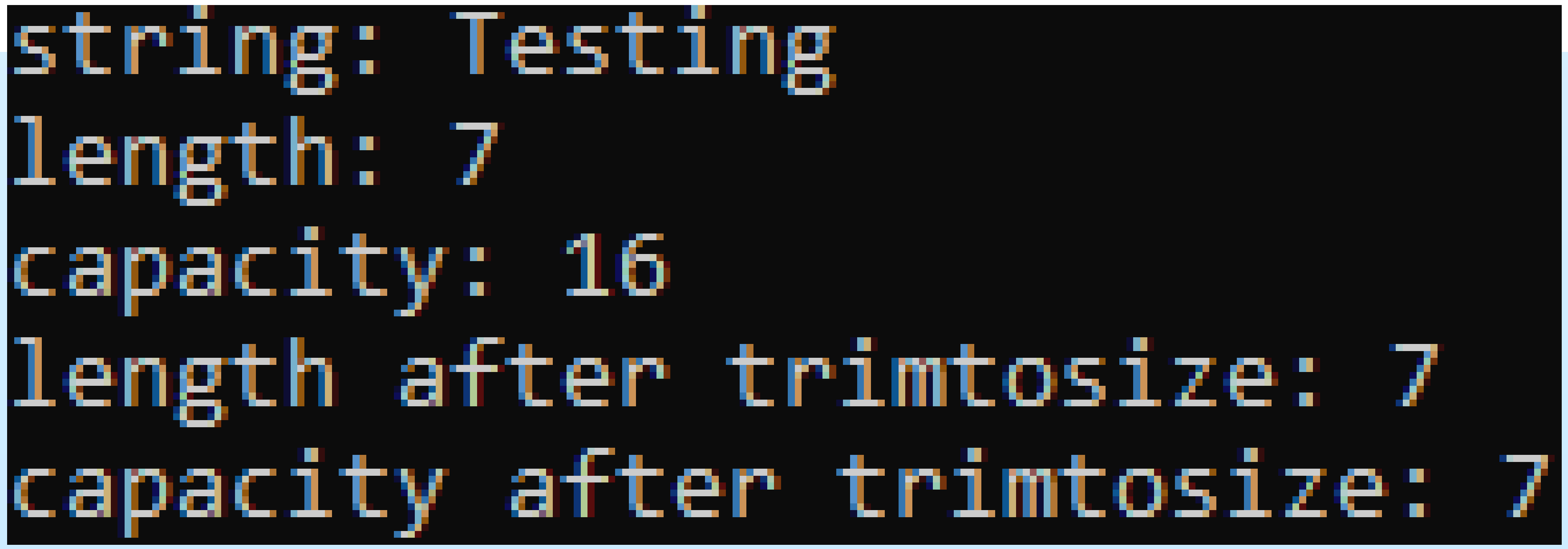
```java
// Demonstrate delete() and deleteCharAt()
class deleteDemo
{
  public static void main(String args[])
  {
    StringBuffer sb = new StringBuffer("This is a test.");

    sb.delete(4, 7);
    System.out.println("After delete: " + sb);

    sb.deleteCharAt(0);
    System.out.println("After deleteCharAt: " + sb);
  }
}
```



```
After delete: This a test.
After deleteCharAt: his a test.
```



```java
public class TrimToSizeExample
{
    public static void main(String[] args)
    {
        StringBuffer sb = new StringBuffer();
        sb.append("Testing");
        System.out.println("string: "+sb);
        int length = sb.length();
        int capacity = sb.capacity();
        System.out.println("length: "+length);
        System.out.println("capacity: "+capacity);
        sb.trimToSize();
        length = sb.length();
        capacity = sb.capacity();
        System.out.println("length after trimtosize: "+length);
        System.out.println("capacity after trimtosize: "+capacity);
     }
}
```

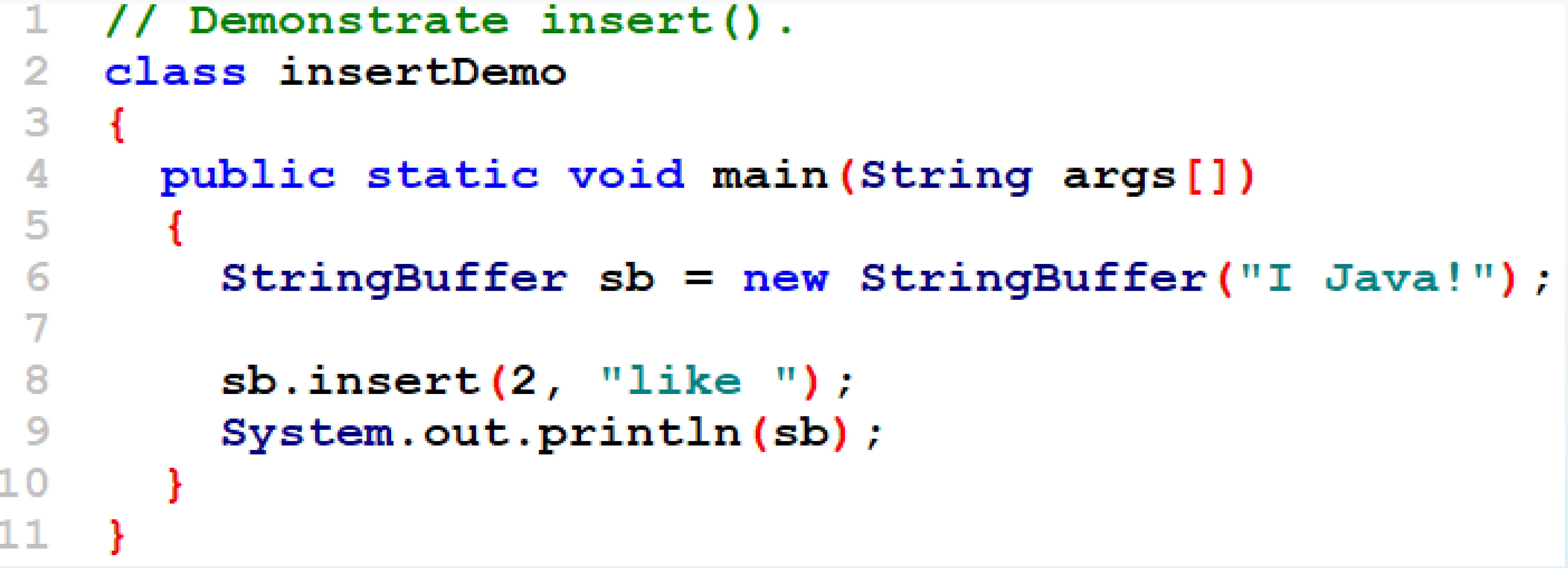


```
string: Testing
length: 7
capacity: 16
length after trimtosize: 7
capacity after trimtosize: 7
```

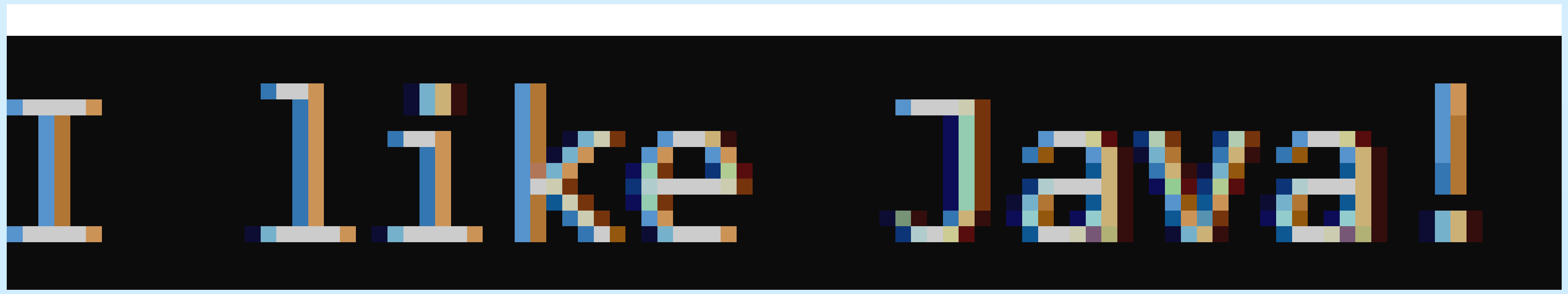
String Handling

- **StringBuffer Class** (Continued …)
  - StringBuffer Methods:

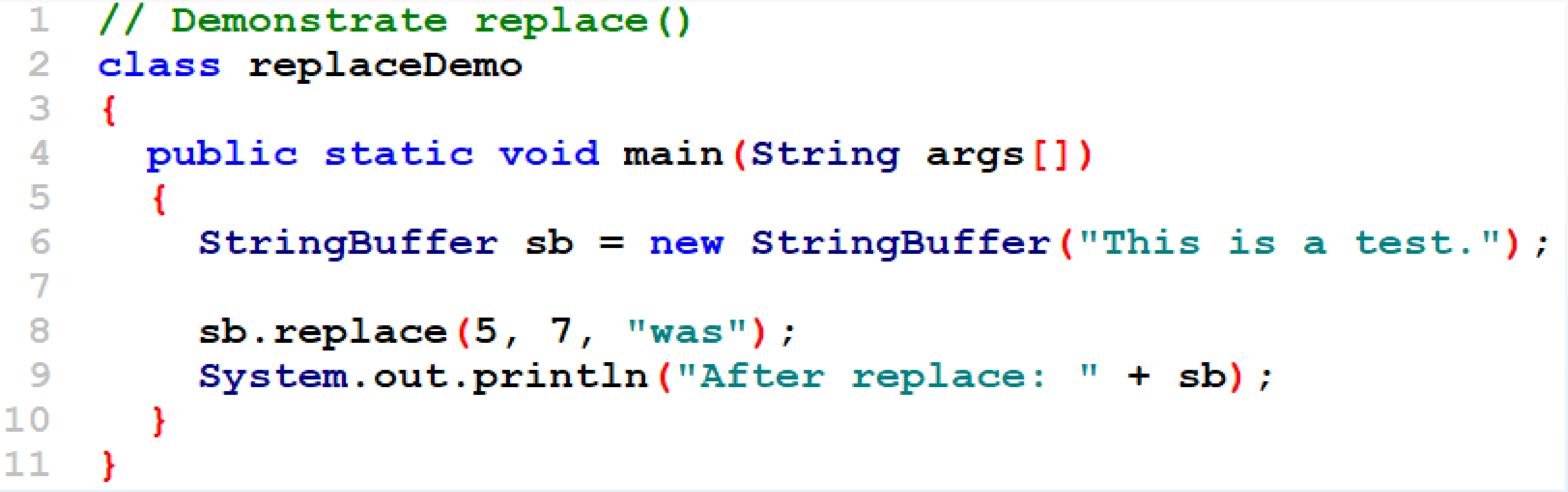
StringBuffer	append (arg)
StringBuffer	append (char[] str, int offset, int len)
StringBuffer	insert (int offset, arg)
StringBuffer	insert (int index, char[] str, int offset, int len)
StringBuffer	replace (int start, int end, String str)
arg = boolean, char, char[], double, float, int, long, string, StringBuffer	



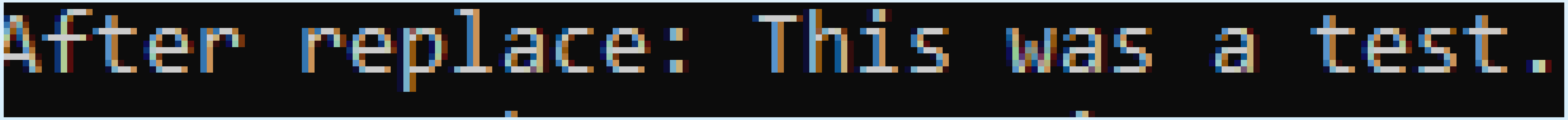
```java
// Demonstrate insert().
class insertDemo
{
  public static void main(String args[])
  {
    StringBuffer sb = new StringBuffer("I Java!");

    sb.insert(2, "like ");
    System.out.println(sb);
  }
}
```



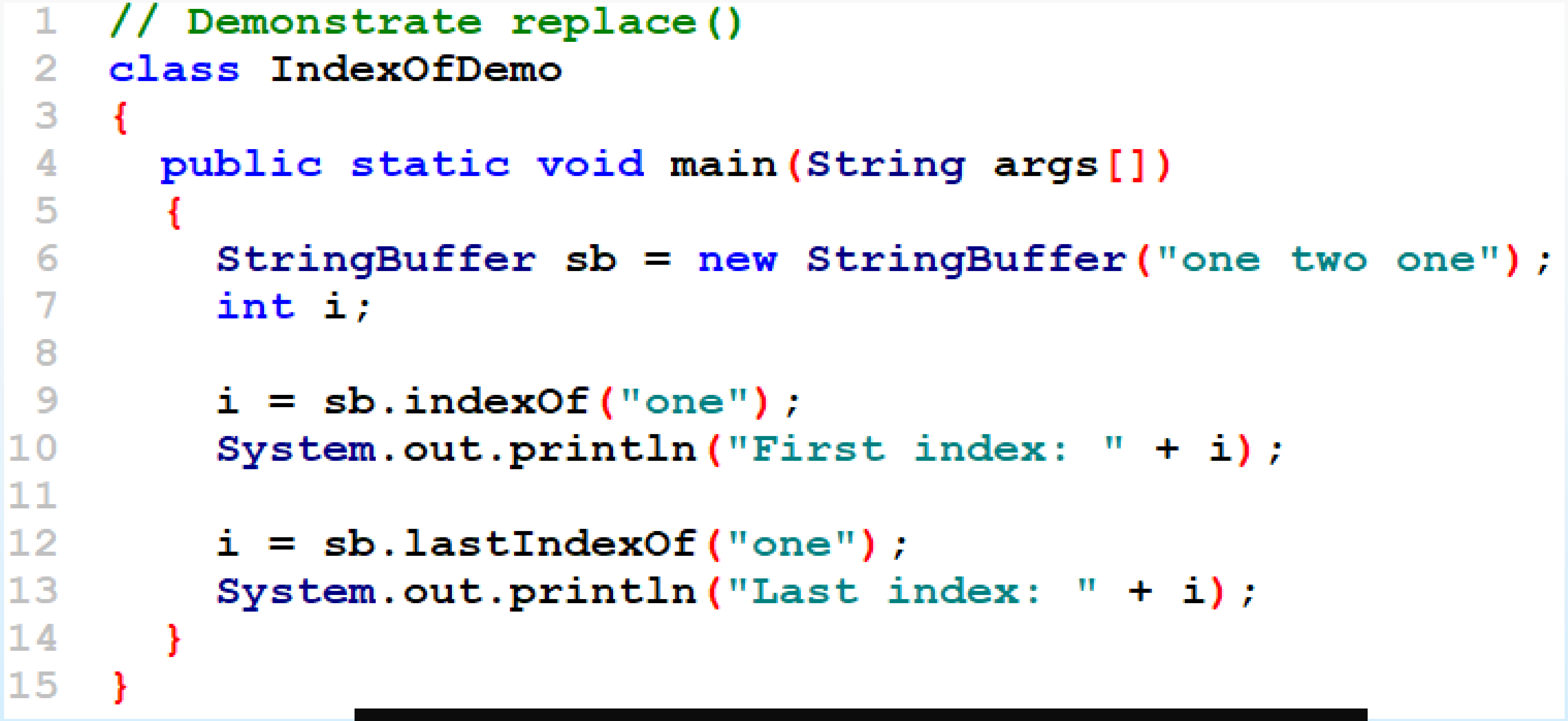
```
I like Java!
```



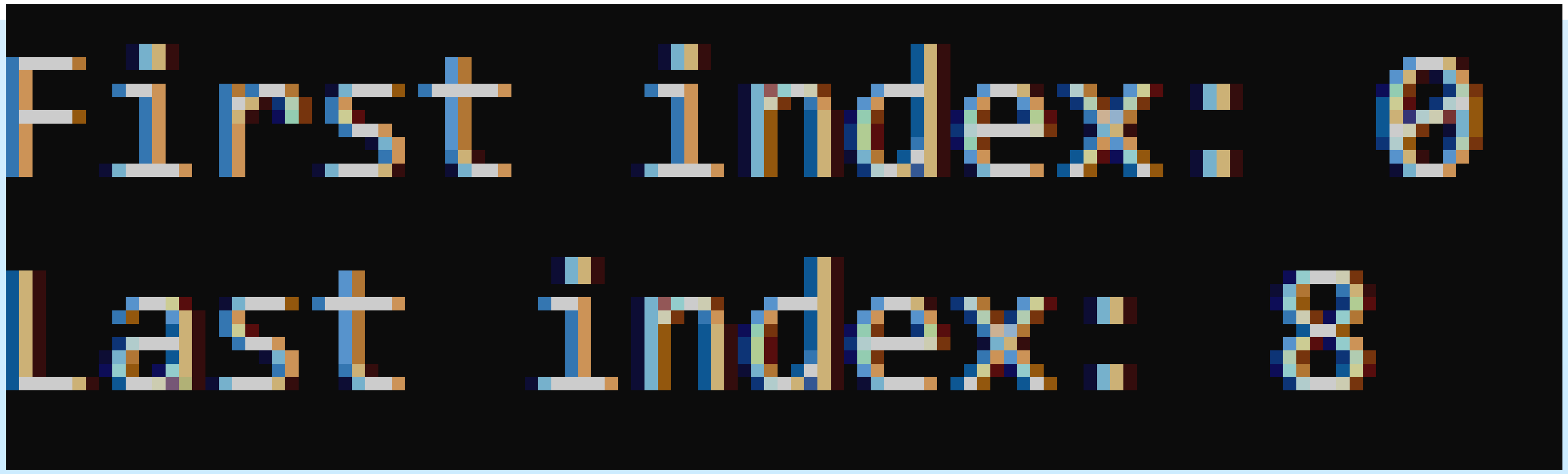
```java
// Demonstrate replace()
class replaceDemo
{
  public static void main(String args[])
  {
    StringBuffer sb = new StringBuffer("This is a test.");

    sb.replace(5, 7, "was");
    System.out.println("After replace: " + sb);
  }
}
```



```
After replace: This was a test.
```



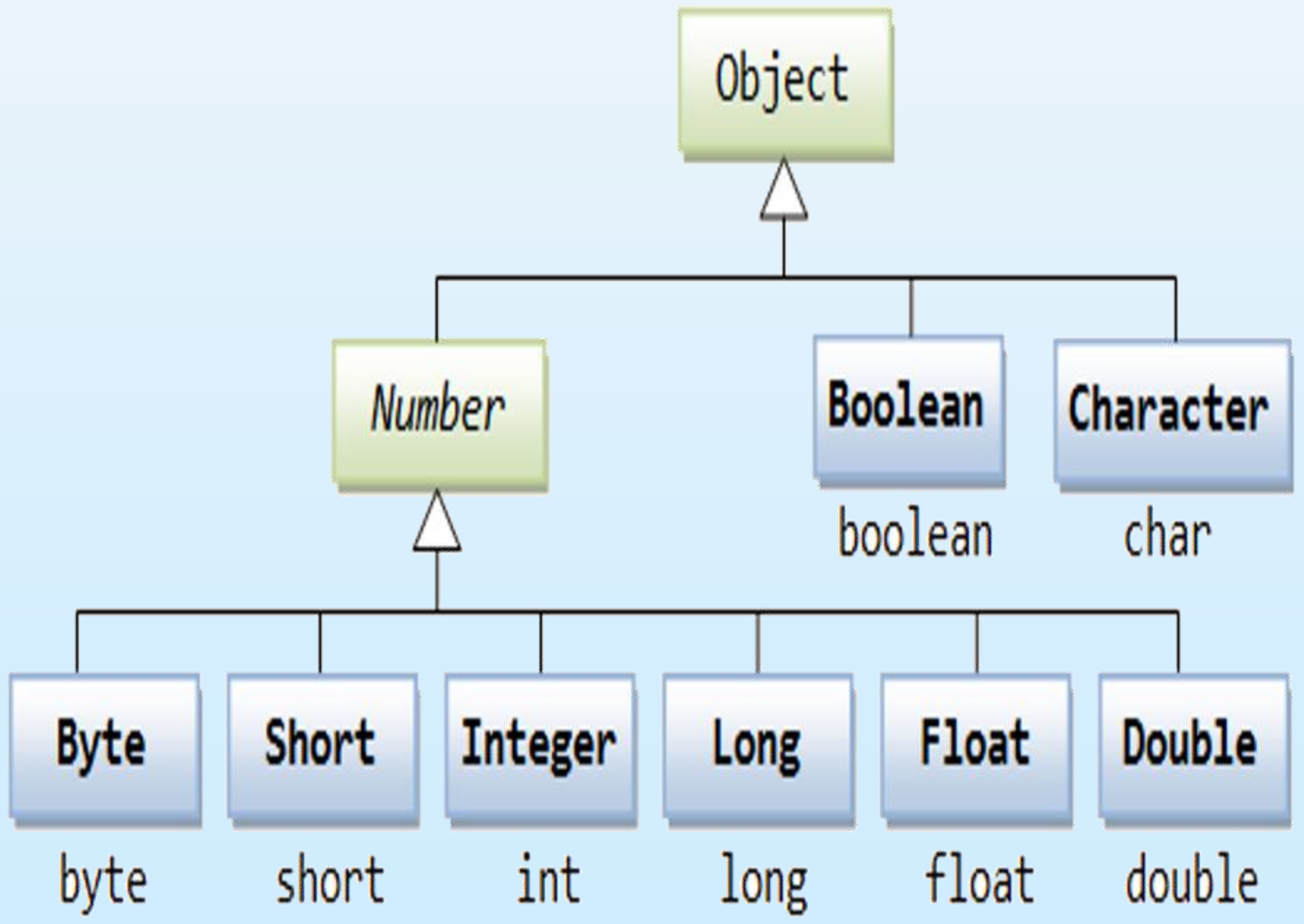
```java
// Demonstrate replace()
class IndexOfDemo
{
  public static void main(String args[])
  {
    StringBuffer sb = new StringBuffer("one two one");
    int i;

    i = sb.indexOf("one");
    System.out.println("First index: " + i);

    i = sb.lastIndexOf("one");
    System.out.println("Last index: " + i);
  }
}
```



```
First index: 0
Last index: 8
```

Wrapper Classes

- **Wrapper Classes**
  - Contained in java.lang package
  - Used for wrapping primitive data into objects and vice versa



Simple Type	Wrapper Class
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
byte	Byte
short	Short

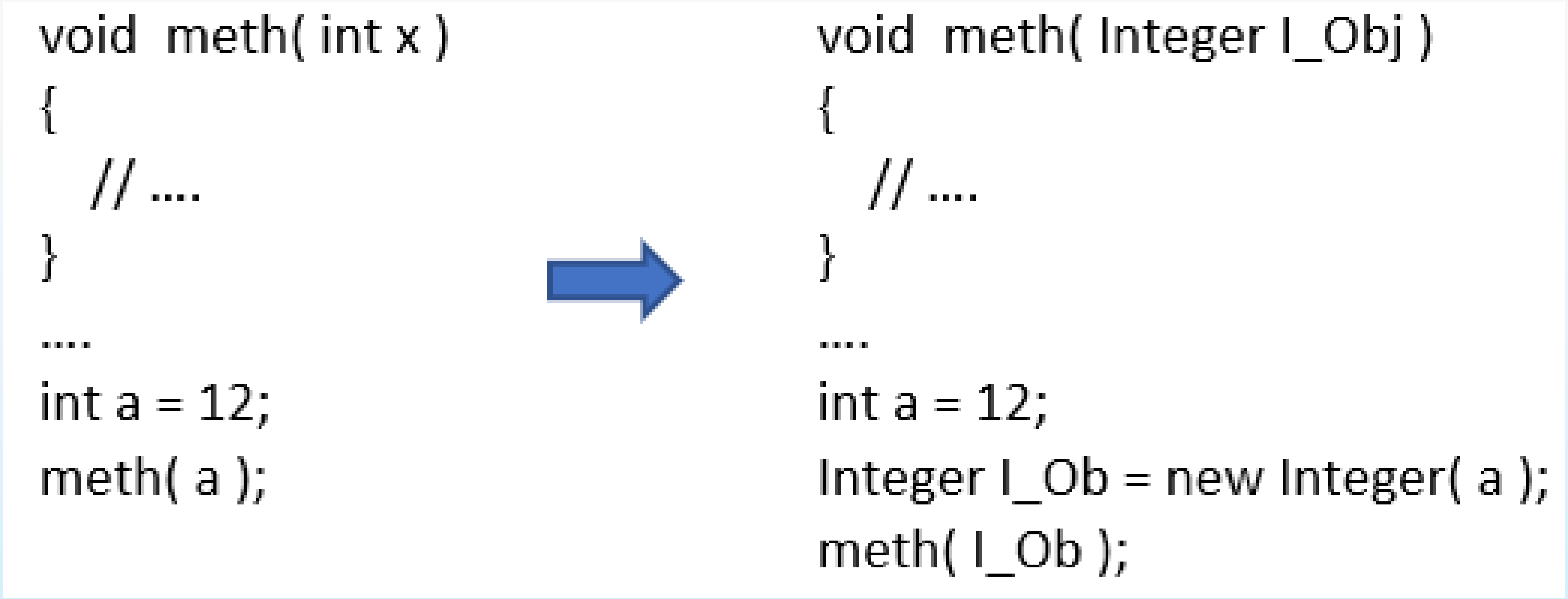
Wrapper Classes

- **Wrapper Classes** (Continued …)
  - Methods for converting primitive numbers to object numbers (using constructor methods)

Method called	**Conversion from**	**Conversion to**
Integer IntObj = new Integer(int)	Primitive int	Integer object
Float FloatObj = new Float(float)	Primitive float	Float object
Double DoubleObj = new Double(double)	Primitive double	Double object
Long LongObj = new Long(long)	Primitive long	Long object
Character CharObj = new Character(char)	Primitive char	Character object
Byte ByteObj = new Byte(byte)	Primitive byte	Byte object
Short ShortObj = new Short(short)	Primitive short	Short object
Boolean BooleanObj = new Boolean(boolean)	Primitive boolean	Boolean object

```
Eg: int x = 10;
Integer I_obj = new Integer( x );
```

Need for wrapper class:



```
void  meth( int x )
{
   // ....
}
....
int a = 12;
meth( a );
```

```
void  meth( Integer I_Obj )
{
   // ....
}
....
int a = 12;
Integer I_Ob = new Integer( a );
meth( I_Ob );
```

Pass by value

Pass by value

Wrapper Classes

- **Wrapper Classes** (Continued …)
  - Methods for converting object numbers to primitive numbers (using typeValue() methods)

Method called	**Conversion from**	**Conversion to**
int intVal = IntObj.intValue()	Integer Object	Primitive int
float floatVal = FloatObj.floatValue()	Float Object	Primitive float
long longVal = LongObj.longValue()	Long Object	Primitive long
double doubleVal = DoubleObj.doubleValue()	Double Object	Primitive double
byte byteVal = ByteObj.byteValue()	Byte Object	Primitive byte
short shortVal = ShortObj.shortValue()	Short Object	Primitive short

Wrapper Classes

- **Wrapper Classes** (Continued …)
  - Methods for converting primitive numbers to string objects (using toString() method)

Method called	**Conversion from**	**Conversion to**
String strVal = Integer.toString(int)	Primitive integer	String
String strVal = Float.toString(float)	Primitive float	String
String strVal = Double.toString(double)	Primitive double	String
String strVal = Long.toString(long)	Primitive long	String
String strVal = Short.toString(short)	Primitive short	String
String strVal = Byte.toString(byte)	Primitive byte	String

Eg:

```
int x = 20;
String s =  Integer.toString( x );  // s → "20"
```

Wrapper Classes

- **Wrapper Classes** (Continued …)
  - Methods for converting string objects to numeric objects (using valueOf() method)

Method called	Conversion from	Conversion to
Integer.valueOf(String)	String	Integer object
Float.valueOf(String)	String	Float object
Long.valueOf(String)	String	Long object
Double.valueOf(String)	String	Double object
Short.valueOf(String)	String	Short object
Byte.valueOf(String)	String	Byte object
Boolean.valueOf(String)	String	Boolean object

Eg:
```
String s = “12”;
Integer I_obj = Integer.valueOf( s );
```

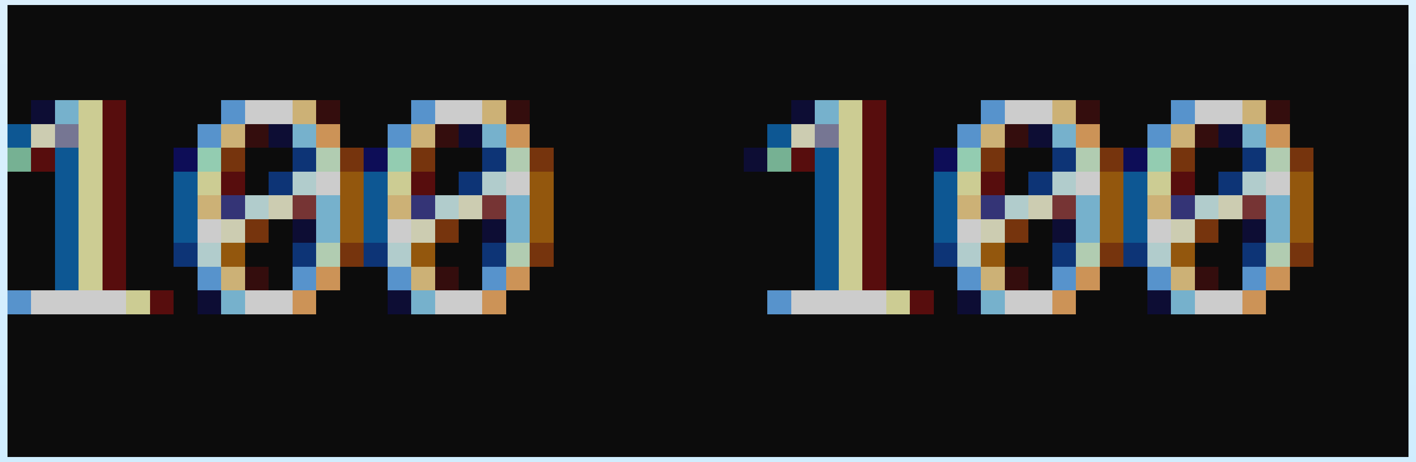
Wrapper Classes

- **Wrapper Classes** (Continued ...)
  - Methods for converting numeric strings to primitive numbers (using parsing method)

Method called	**Conversion from**	**Conversion to**
int intVal = Integer.parseInt(String)	String	Primitive integer
long longVal = Long.parseLong(String)	String	Primitive long
float floatVal = Float.parseFloat(String)	String	Primitive float
double doubleVal = Double.parseDouble(String)	String	Primitive double
byte byteVal = Byte.parseByte(String)	String	Primitive byte
short shortVal = Short.parseShort(String)	String	Primitive short
boolean booleanVal = Boolean.parseBoolean(String)	String	Primitive boolean

Eg:
```
String s = “12.5”;
float f = Float.parseFloat( s );
```

```java
// Demonstrate a type wrapper.
class Wrap
{
  public static void main(String args[])
  {

    Integer iOb = new Integer(100);

    int i = iOb.intValue();

    System.out.println(i + " " + iOb); // displays 100 100
  }
}
```



```
100 100
```

Autoboxing and Unboxing

Autoboxing

- Automatic conversion of primitive type into corresponding wrapper class object (boxed into wrapper class)
  - Occurs when an object is expected and primitive type is available
  - Uses valueOf() method to convert primitive to Object

Examples:

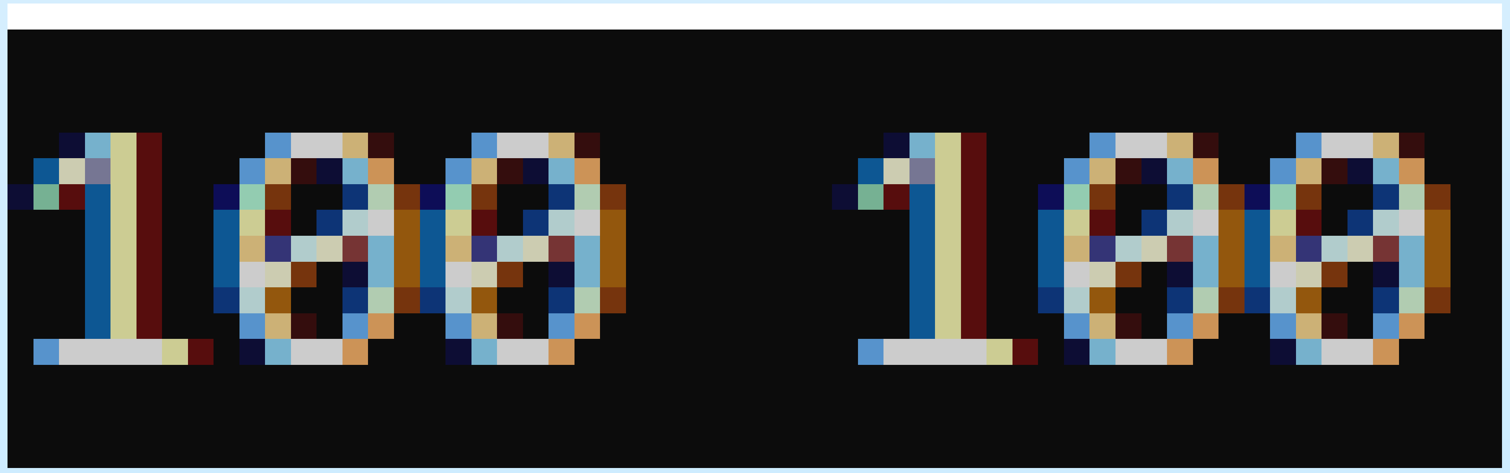
1. Integer iObject = 3;
2. public static void show (Integer iParam)

```
        {
                System.out.println (iParam);
        }
```

In calling method:

```
        show(3);
```

```java
// Demonstrate a autoboxing/unboxing.
class AutoBox
{
  public static void main(String args[])
  {

    Integer iOb = 100; // autobox an int

    int i = iOb; // auto-unbox

    System.out.println(i + " " + iOb);
  }
}
```



```
100 100
```

Autoboxing and Unboxing

- **Unboxing**
  - Automatic conversion of object into corresponding primitive type (unboxed from wrapper class)
    - Occurs when an primitive type is expected and object is available
    - Uses intValue(), doubleValue() etc. to get primitive value from Object

Examples:

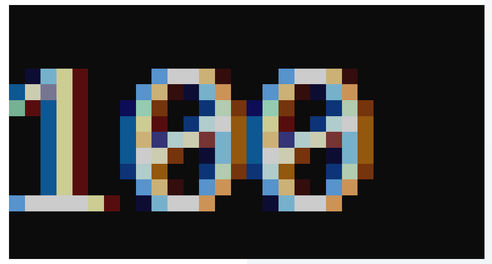
1. int n = iObject;
2. public static Integer show (Integer iParam)

```
        {
                System.out.println (iParam);
                return iParam;
        }
```

In calling method:

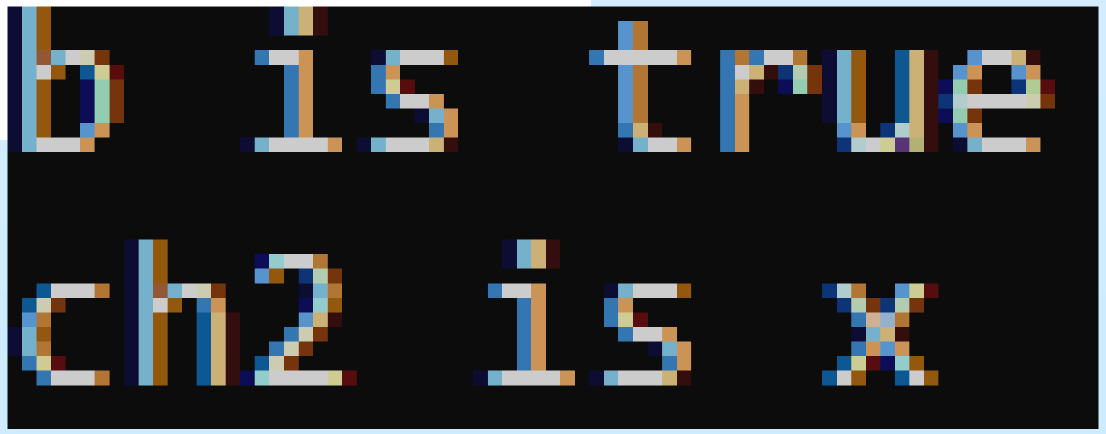
```
        int result = show(3);
```

```java
// Autoboxing/unboxing takes place with
// method parameters and return values.

class AutoBox2
{
  static int m(Integer v)
  {
    return v ; // auto-unbox to int
  }

  public static void main(String args[])
  {

    Integer iOb = m(100);

    System.out.println(iOb);
  }
}
```



```
100
```

```java
// Autoboxing/unboxing a Boolean and Character.
 class AutoBox5
{
  public static void main(String args[])
   {

     // Autobox/unbox a boolean.
     Boolean b = true;

     // Below, b is auto-unboxed when used in
     // a conditional expression, such as an if.
     if(b) System.out.println("b is true");

     // Autobox/unbox a char.
     Character ch = 'x'; // box a char
     char ch2 = ch; // unbox a char

     System.out.println("ch2 is " + ch2);
   }
}
```

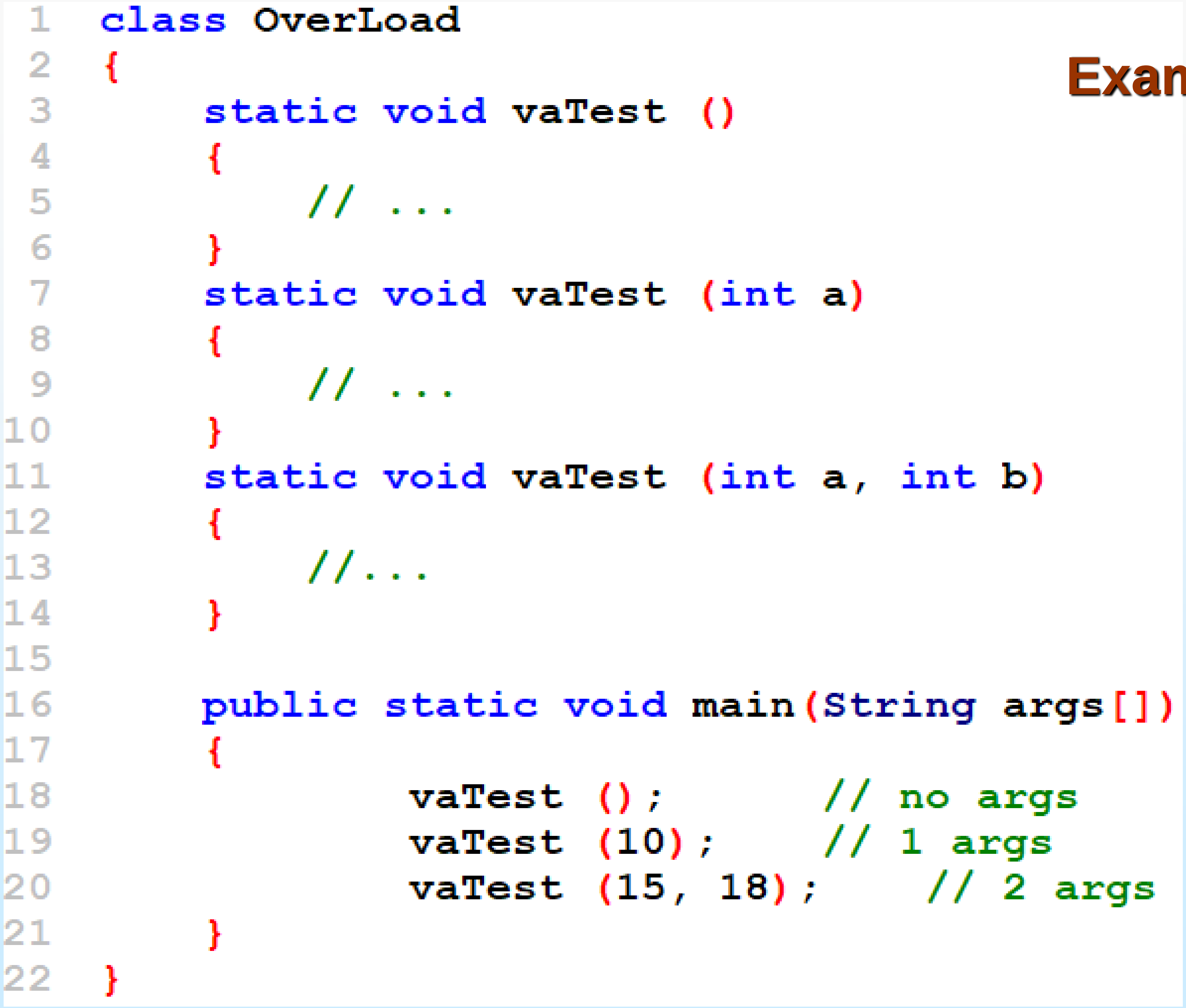


```
b is true
ch2 is x
```

Variable-Length Arguments

- **Varargs**
  - New technique introduced in JDK 5
  - Earlier techniques:
    - If the maximum number of arguments is small and known, create overloaded versions of the method one for each way the method could be handled
      - Example 1
    - If the maximum number of arguments is large and unknown, put the arguments in an array and pass the array to the method
      - Example 2

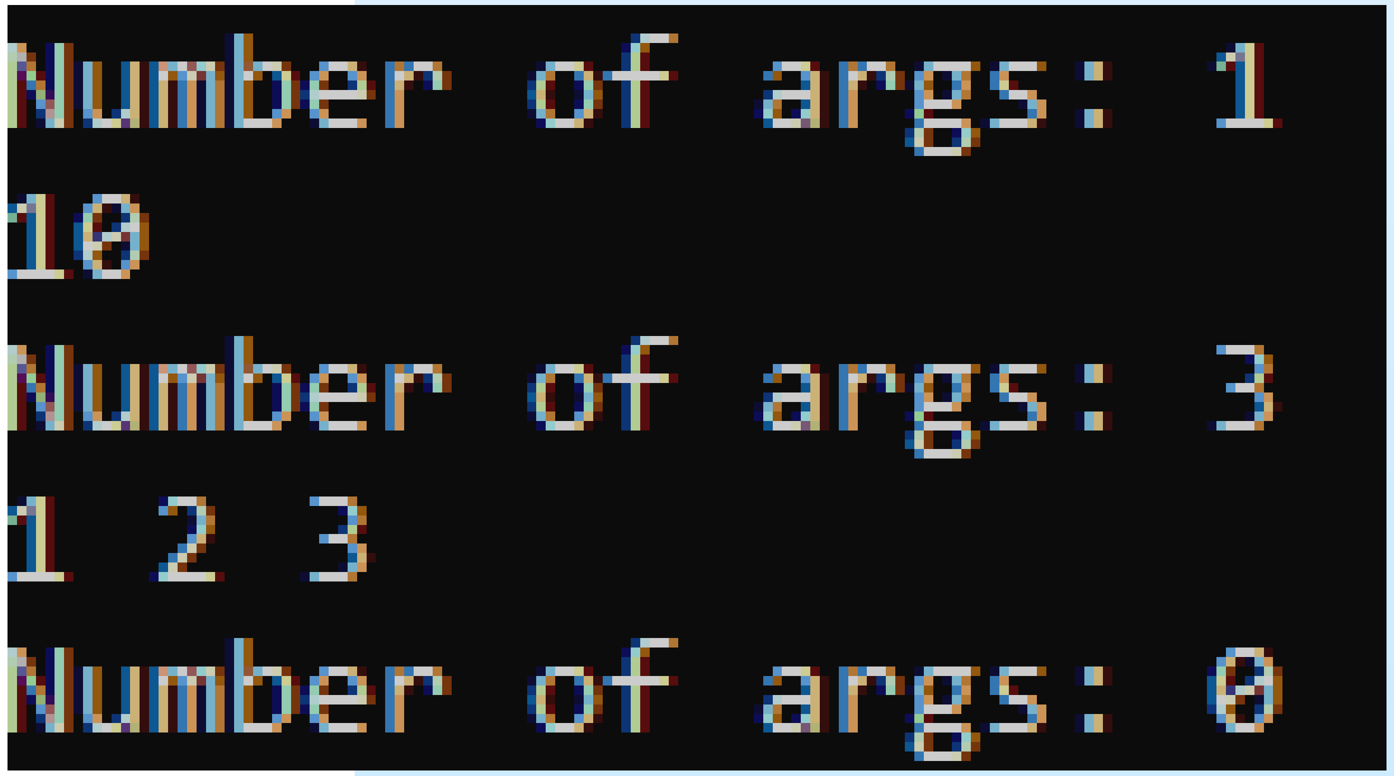
Example-1



```java
class OverLoad
{
    static void vaTest ()
    {
        // ...
    }
    static void vaTest (int a)
    {
        // ...
    }
    static void vaTest (int a, int b)
    {
        //...
    }

    public static void main(String args[])
    {
            vaTest ();      // no args
            vaTest (10);    // 1 args
            vaTest (15, 18);    // 2 args
    }
}
```

Example-2

```java
// Use an array to pass a variable number of
// arguments to a method.
class PassArray
{
  static void vaTest(int v[])
  {
    System.out.println("Number of args: "+v.length);

    for(int x : v)
      System.out.print(x + " ");

    System.out.println();
  }
  public static void main(String args[])
  {
    int n1[] = { 10 };
    int n2[] = { 1, 2, 3 };
    int n3[] = { };

    vaTest(n1); // 1 arg
    vaTest(n2); // 3 args
    vaTest(n3); // no args
  }
}
```

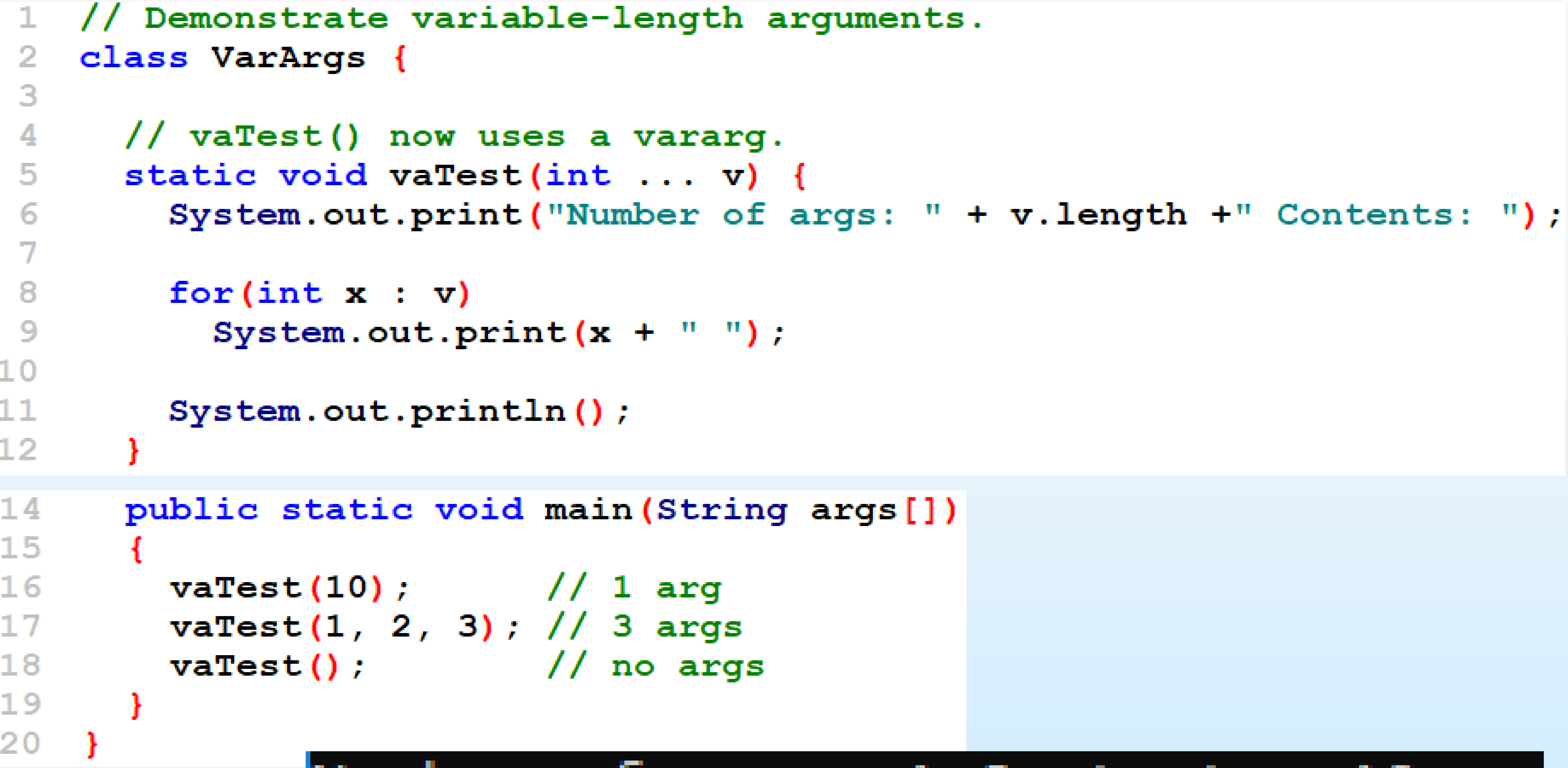


```
Number of args: 1
10
Number of args: 3
1 2 3
Number of args: 0
```

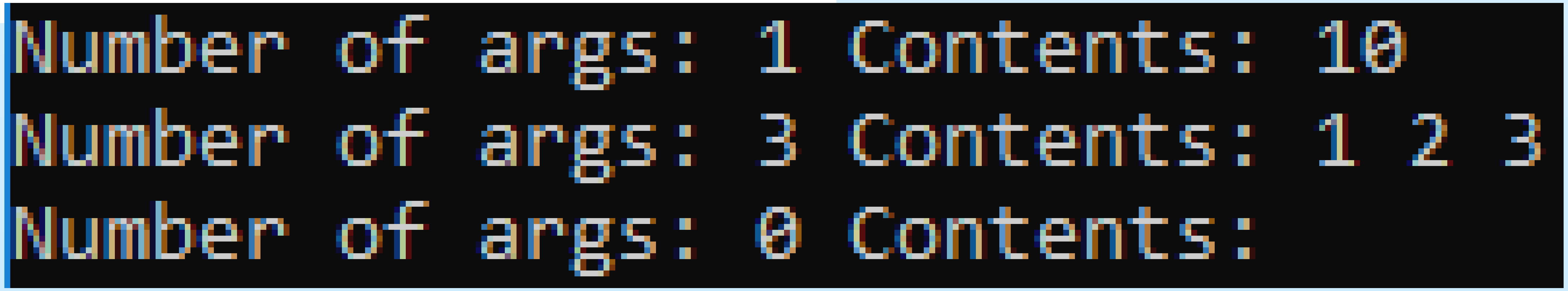
Variable-Length Arguments

- **Varargs** (Continued …)
  - A variable-length argument is specified by three dots (…)

```
class VarArgs
{
        static void vaTest (int … v)
        {

        }

        public static void main(String args[])
        {
                vaTest ();          // no args
                vaTest (10);        // 1 arg
                vaTest (15, 18);    // 2 args
        }
}
```



```java
// Demonstrate variable-length arguments.
class VarArgs {

  // vaTest() now uses a vararg.
  static void vaTest(int ... v) {
    System.out.print("Number of args: " + v.length +" Contents: ");

    for(int x : v)
      System.out.print(x + " ");

    System.out.println();
  }
  public static void main(String args[])
  {
    vaTest(10);       // 1 arg
    vaTest(1, 2, 3); // 3 args
    vaTest();         // no args
  }
}
```



```
Number of args: 1 Contents: 10
Number of args: 3 Contents: 1 2 3
Number of args: 0 Contents:
```

Variable-Length Arguments

- **Varargs** (Continued …)
  - Normal parameters can be used with variable-length arguments
    - Normal parameters should precede variable-length arguments
  - There must be only one variable-length parameter

Examples:

```
void vaTest (int a, float b, char c, int … vals);        // valid
void vaTest (int a, int … vals, float b, char c);        // invalid
void vaTest (int a, int … v1, double … v2);              // invalid
```

Variable-Length Arguments

- **Varargs** (Continued …)
  - Varargs methods can be overloaded

```
class VarArgs {
        static void vaTest (int ... v) { }
        static void vaTest (boolean ... v) { }
        static void vaTest (double ... v) { }

        public static void main(String args[])
        {
                vaTest (1, 2, 3);
                vaTest (true, false, false);
                vaTest (12.75, 17.45, -34.6, 23.91);
        }
}
```

Variable-Length Arguments

- **Varargs** (Continued …)
  - Overloaded Varargs methods can lead to ambiguity

Case 1:

```
class VarArgs {
        static void vaTest (int ... v) { }
        static void vaTest (boolean ... v) { }

        public static void main(String args[])
        {
                vaTest (1, 2, 3);
                vaTest (true, false, false);
                vaTest ();              // ambiguous
        }
}
```

Since the *vararg* parameter can be empty, the last call can be translated into a call to either of the two methods

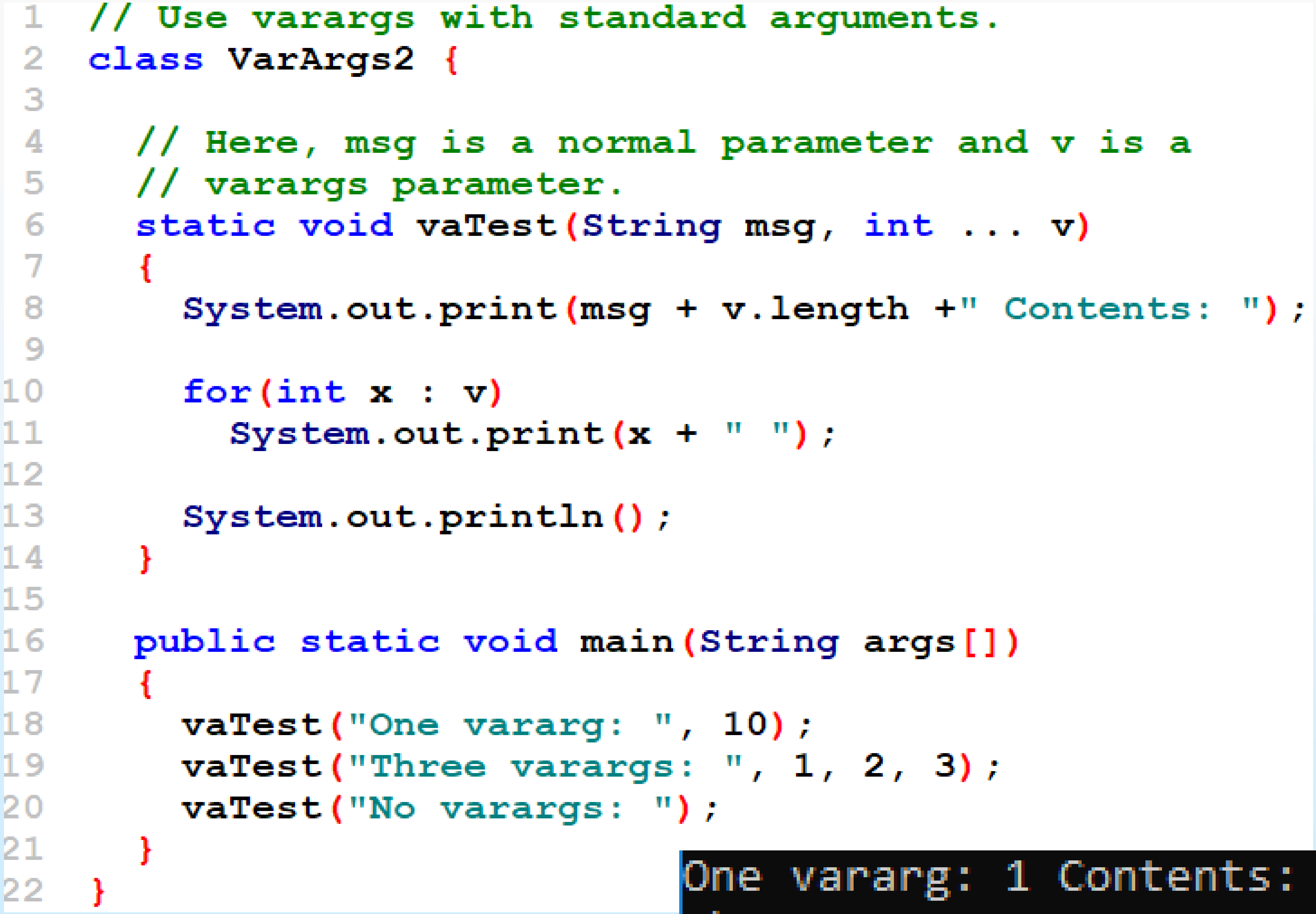
Variable-Length Arguments

- **Varargs** (Continued …)
  - Overloaded Varargs methods can lead to ambiguity

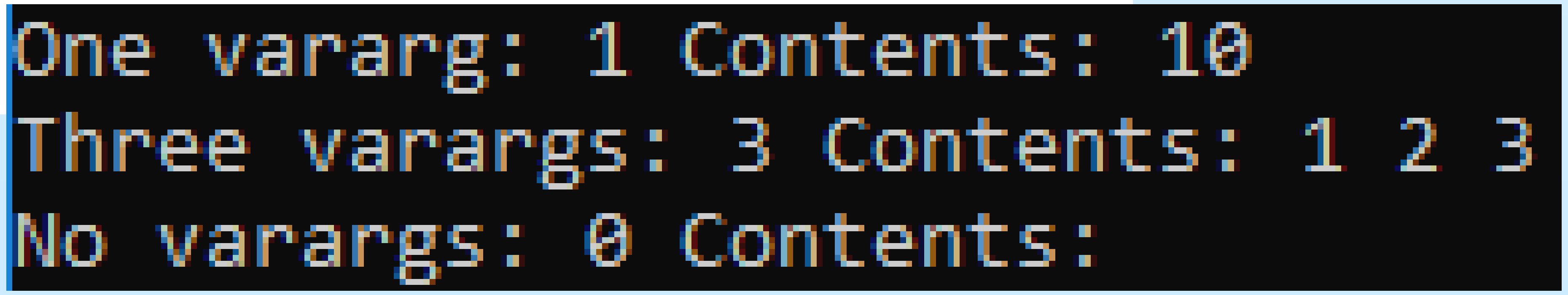
    Case 2:

```
class VarArgs {
        static void vaTest (int ... v) { }
        static void vaTest (int) { }

        public static void main(String args[])
        {
                vaTest (1);          // ambiguous
        }
}
```

Compiler cannot resolve the issue of whether to translate the method call into first method or the second

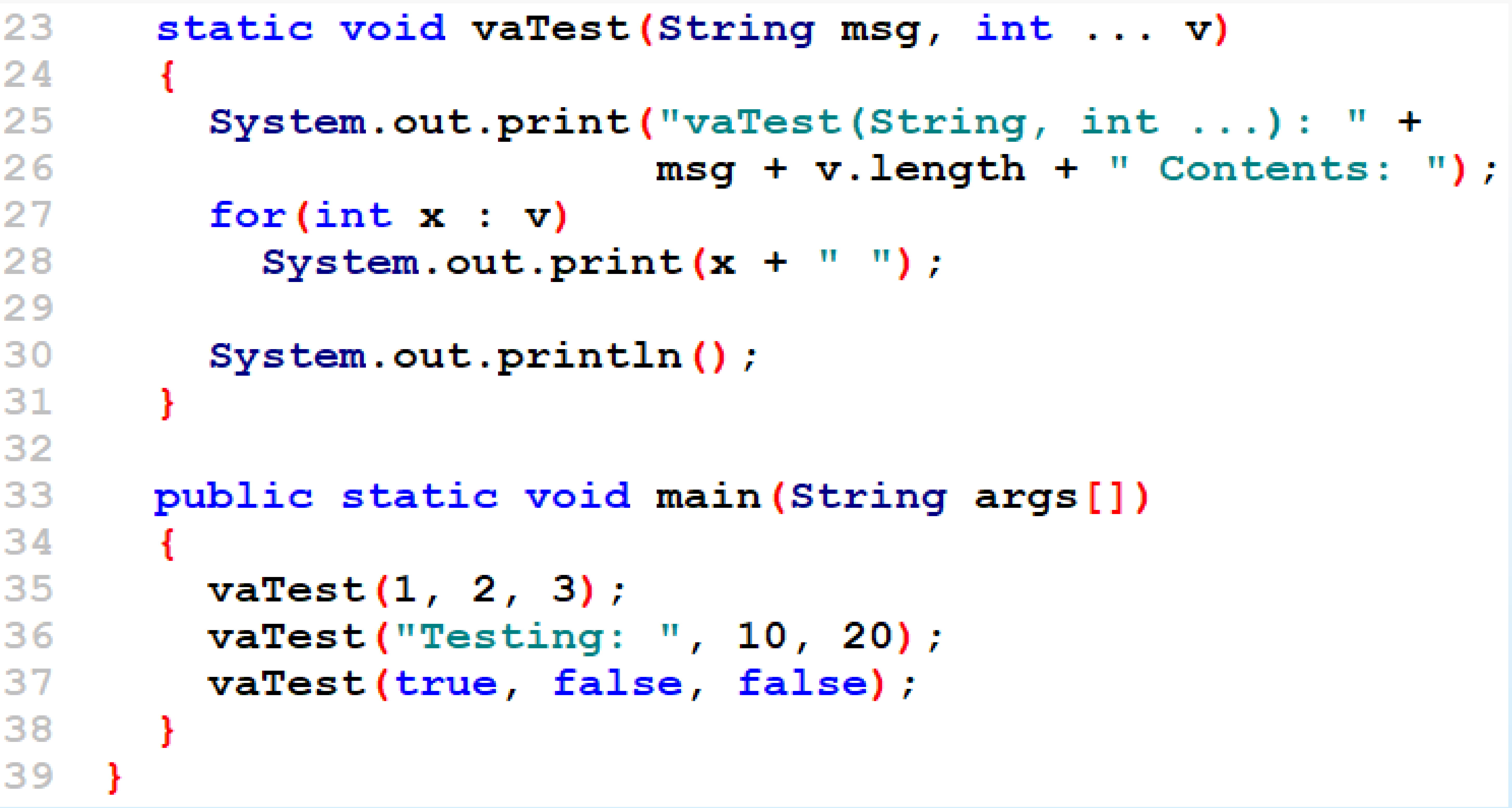


```java
// Use varargs with standard arguments.
class VarArgs2 {

  // Here, msg is a normal parameter and v is a
  // varargs parameter.
  static void vaTest(String msg, int ... v)
  {
    System.out.print(msg + v.length +" Contents: ");

    for(int x : v)
      System.out.print(x + " ");

    System.out.println();
  }

  public static void main(String args[])
  {
    vaTest("One vararg: ", 10);
    vaTest("Three varargs: ", 1, 2, 3);
    vaTest("No varargs: ");
  }
}
```

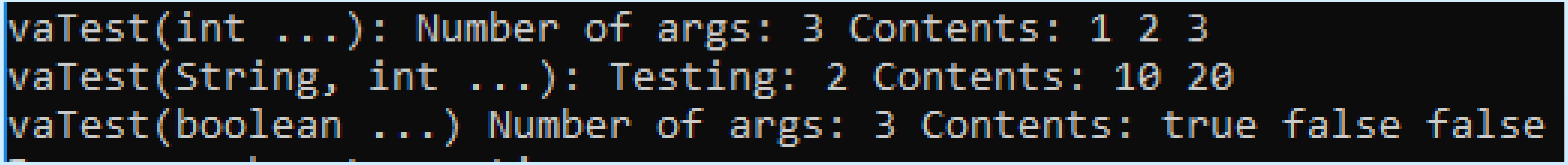


```
One vararg: 1 Contents: 10
Three varargs: 3 Contents: 1 2 3
No varargs: 0 Contents:
```

```java
// Varargs and overloading.
class VarArgs3
{
  static void vaTest(int ... v)
  {
    System.out.print("vaTest(int ...): " +
                     "Number of args: " + v.length +" Contents: ");

    for(int x : v)
      System.out.print(x + " ");
     System.out.println();
  }

  static void vaTest(boolean ... v)
  {
    System.out.print("vaTest(boolean ...) " +
                     "Number of args: " + v.length +" Contents: ");
     for(boolean x : v)
      System.out.print(x + " ");
    System.out.println();
  }
```

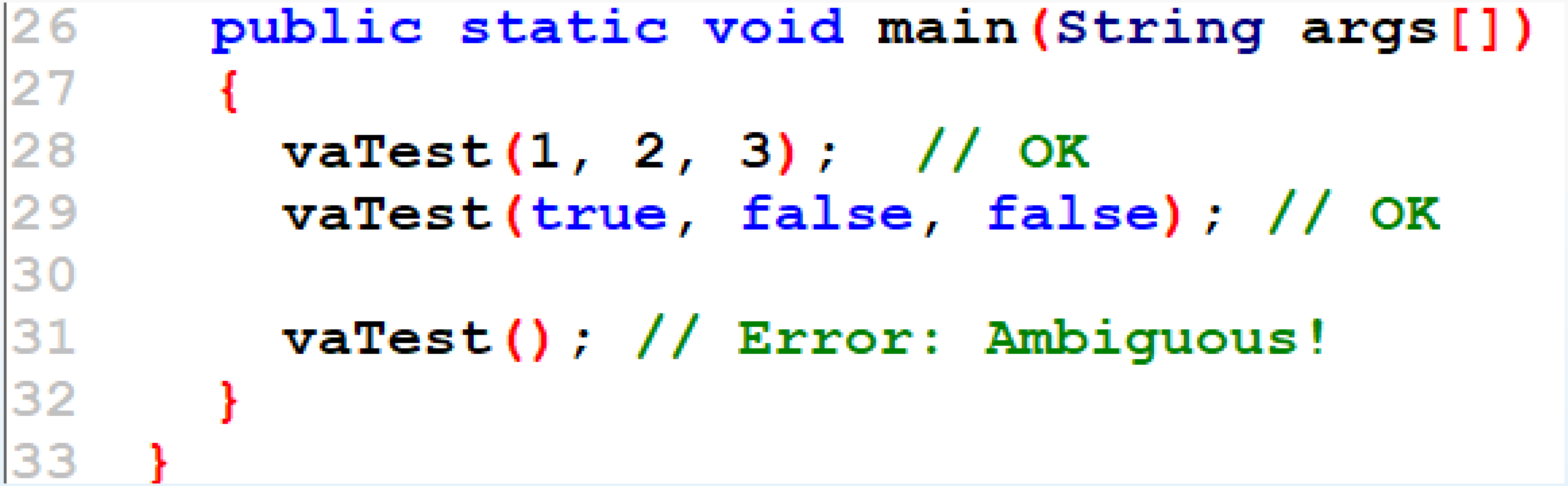


```java
    static void vaTest(String msg, int ... v)
    {
      System.out.print("vaTest(String, int ...): " +
                       msg + v.length + " Contents: ");
      for(int x : v)
        System.out.print(x + " ");

      System.out.println();
    }

    public static void main(String args[])
    {
      vaTest(1, 2, 3);
      vaTest("Testing: ", 10, 20);
      vaTest(true, false, false);
    }
  }
```



```
vaTest(int ...): Number of args: 3 Contents: 1 2 3
vaTest(String, int ...): Testing: 2 Contents: 10 20
vaTest(boolean ...) Number of args: 3 Contents: true false false
```

```java
class VarArgs4 {

  static void vaTest(int ... v)
  {
    System.out.print("vaTest(Integer ...): " +
                     "Number of args: " + v.length + " Contents: ");

    for(int x : v)
      System.out.print(x + " ");

    System.out.println();
  }

  static void vaTest(boolean ... v)
  {
    System.out.print("vaTest(boolean ...) " +
                     "Number of args: " + v.length + " Contents: ");

    for(boolean x : v)
      System.out.print(x + " ");

    System.out.println();
  }
```



```java
  public static void main(String args[])
  {
    vaTest(1, 2, 3);  // OK
    vaTest(true, false, false); // OK

    vaTest(); // Error: Ambiguous!
  }
}
```

The End