Introduction to Java

Dr. Jafar Tanha
Univerity of Tabriz

Why Java?

- It’s the current “hot” language
- It’s almost entirely object-oriented
- It has a vast library of predefined objects and operations
- It’s more platform independent
  - this makes it great for Web programming
- It’s more secure
- It isn’t C++

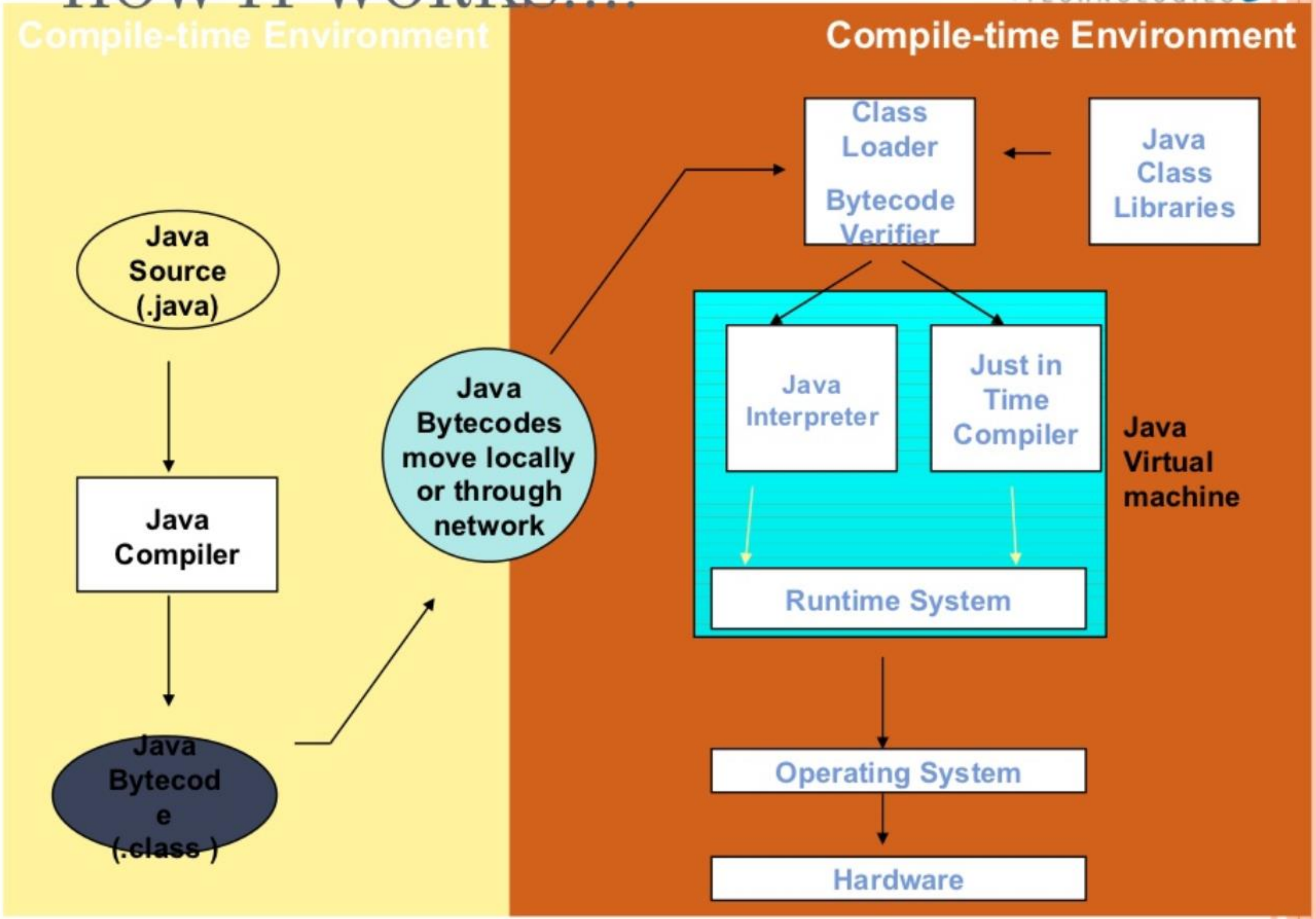
JAVA - GENERAL

- Java has some interesting features:
  - automatic type checking,
  - automatic garbage collection,
  - simplifies pointers; no directly accessible pointer to memory,
  - simplified network access,
  - multi-threading!

Characteristics of Java

- Java is simple
- Java is object-oriented
- Java is distributed
- Java is interpreted
- Java is robust
- Java is secure
- Java is architecture-neutral
- Java is portable
- Java’s performance
- Java is multithreaded
- Java is dynamic

HOW IT WORKS…!


Compile-time Environment
Compile-time Environment
Java Source (.java)
Java Compiler
Java Bytecode (.class)
Java Bytecodes move locally or through network
Class Loader
Bytecode Verifier
Java Class Libraries
Java Interpreter
Just in Time Compiler
Runtime System
Java Virtual machine
Operating System
Hardware

- Java is independent only for one reason:
  - Only depends on the Java Virtual Machine (JVM),
  - code is compiled to *bytecode*, which is interpreted by the resident JVM,
  - JIT (just in time) compilers attempt to increase speed.

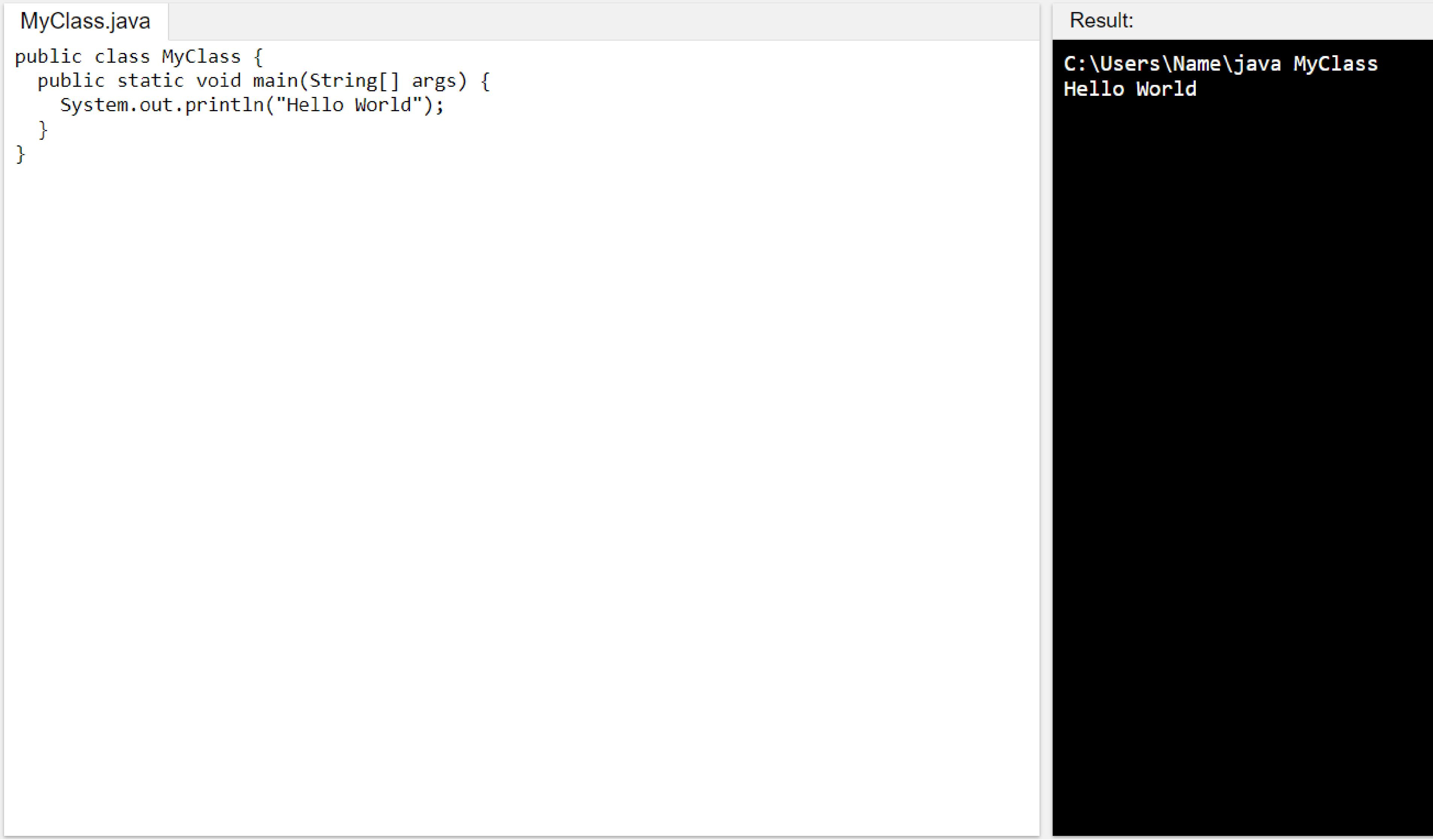
Java Virtual Machine

- The .class files generated by the compiler are not executable binaries
  - so Java combines compilation and interpretation
- Instead, they contain “byte-codes” to be executed by the Java Virtual Machine
  - other languages have done this, e.g. UCSD Pascal
- This approach provides platform independence, and greater security

HelloWorld (standalone)

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- Note that String is built in
- println is a member function for the System.out class


MyClass.java

```java
public class MyClass {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Result:

```
C:\Users\Name\java MyClass
Hello World
```

Java Quickstart

In Java, every application begins with a class name, and that class must match the filename.

Let's create our first Java file, called MyClass.java, which can be done in any text editor (like Notepad).

The file should contain a "Hello World" message, which is written with the following code:

MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Run example »

Save the code in Notepad as "MyClass.java". Open Command Prompt (cmd.exe), navigate to the directory where you saved your file, and type "javac MyClass.java"

```
C:\Users\Your Name>javac MyClass.java
```

This will compile your code. If there are no errors in the code, the command prompt will take you to the next line. Now, type "java MyClass" to run the file:

```
C:\Users\Your Name>java MyClass
```

The output should read:

```
Hello World
```

Run example »

The main Method

The `main()` method is required and you will see it in every Java program:

```
public static void main(String[] args)
```

System.out.println()

Inside the `main()` method, we can use the `println()` method to print a line of text to the screen:

```
public static void main(String[] args) {
  System.out.println("Hello World");
}
```

Run example »

Comments are almost like C++

- `/* This kind of comment can span multiple lines */`

- `// This kind is to the end of the line`

-
  ```
  /**
    * This kind of comment is a special
    * ‘javadoc’ style comment
    */
  ```

Java Comments

❮ Previous Next ❯

Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code

Single-line comments start with two forward slashes (`//`).

Any text between `//` and the end of the line is ignored by Java (will not be executed).

This example uses a single-line comment before a line of code:

Example

```
// This is a comment
System.out.println("Hello World");
```

Run example »

Java Multi-line Comments

Multi-line comments start with /* and ends with */.

Any text between /* and */ will be ignored by Java.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/* The code below will print the words Hello World
to the screen, and it is amazing */
System.out.println("Hello World");
```

Run example »

Java Identifiers

All Java **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code.

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs
- Names must begin with a letter
- Names should start with a lowercase letter and it cannot contain whitespace
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like Java keywords, such as `int` or `boolean`) cannot be used as names

Java Data Types

As explained in the previous chapter, a variable in Java must be a specified data type:

Example

```
int myNum = 5;               // Integer (whole number)
float myFloatNum = 5.99f;    // Floating point number
char myLetter = 'D';         // Character
boolean myBool = true;       // Boolean
String myText = "Hello";     // String
```

Run example »

Primitive data types are like C

Main data types are:

- `int, double, boolean, char`
- Also have `byte, short, long, float`
- `boolean` has values `true` and `false`
- Declarations look like C, for example,
  - `double x, y;`
  - `int count = 0;`

Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Booleans

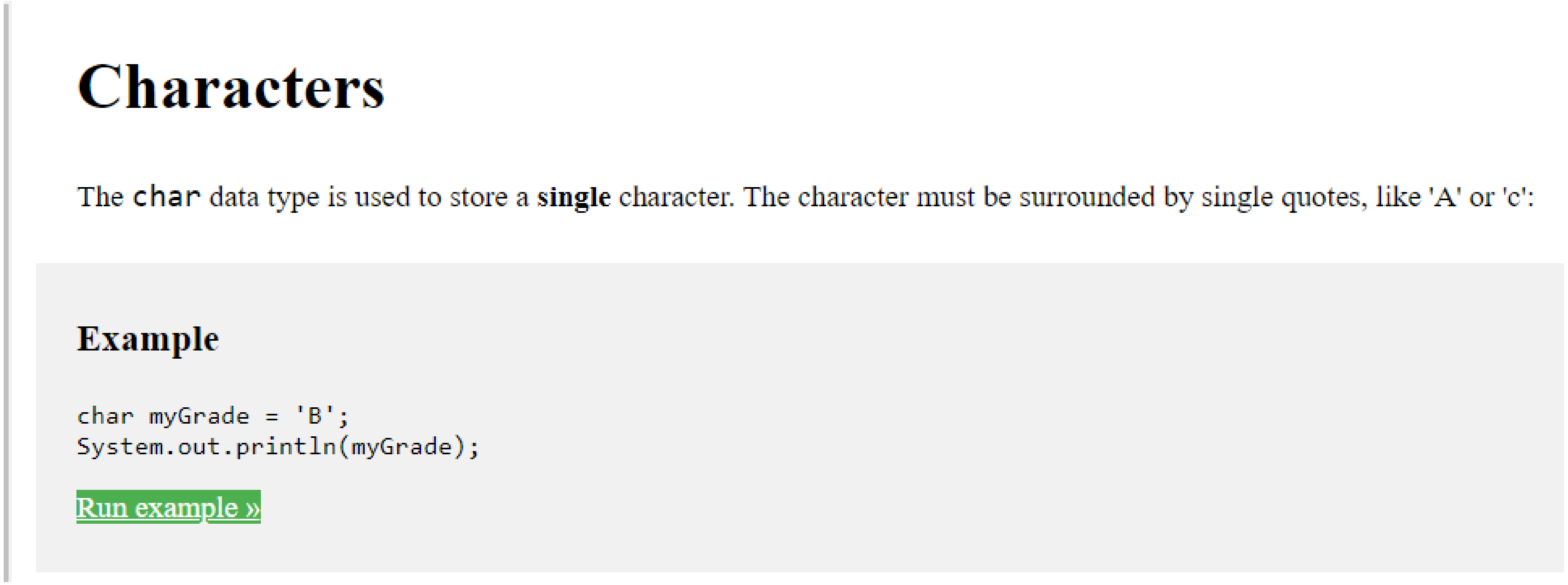
A boolean data type is declared with the `boolean` keyword and can only take the values `true` or `false`:

Example

```
boolean isJavaFun = true;
boolean isFishTasty = false;
System.out.println(isJavaFun);     // Outputs true
System.out.println(isFishTasty);   // Outputs false
```

Run example »

Boolean values are mostly used for conditional testing, which you will learn more about in a later chapter.



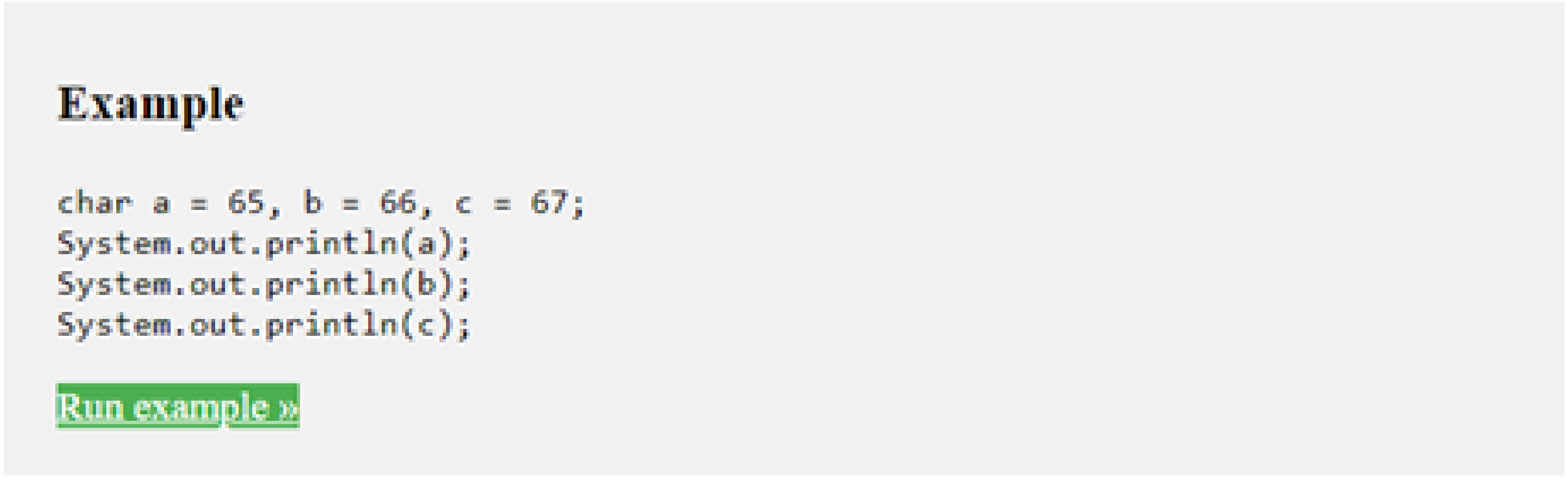
Characters

The `char` data type is used to store a **single** character. The character must be surrounded by single quotes, like 'A' or 'c':

Example

```
char myGrade = 'B';
System.out.println(myGrade);
```

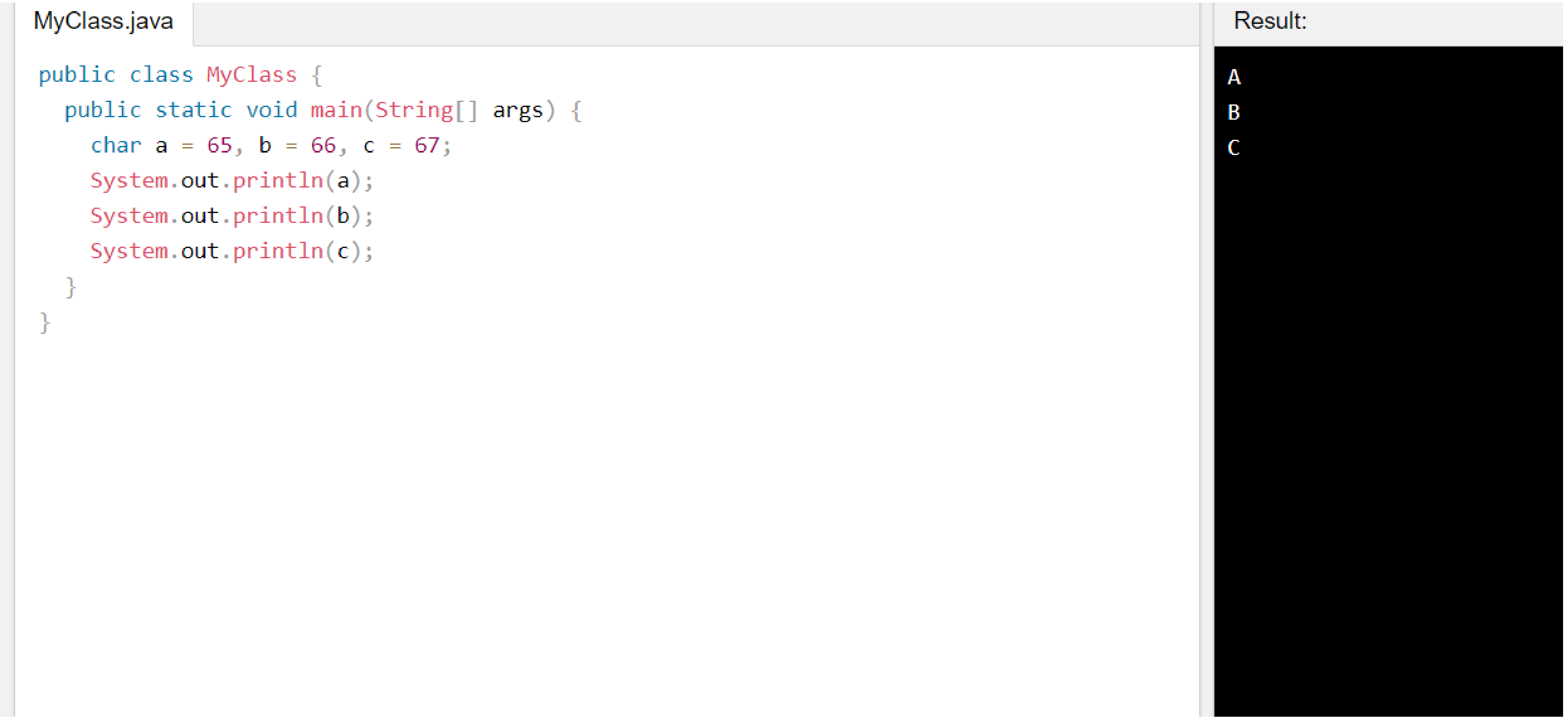
Run example »



Example

```
char a = 65, b = 66, c = 67;
System.out.println(a);
System.out.println(b);
System.out.println(c);
```

Run example »



MyClass.java

```java
public class MyClass {
  public static void main(String[] args) {
    char a = 65, b = 66, c = 67;
    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
  }
}
```

Result:

```
A
B
C
```

Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
  `byte -> short -> char -> int -> long -> float -> double`

- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
  `double -> float -> long -> int -> char -> short -> byte`

Widening Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

Example

```
public class MyClass {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

Example

```
public class MyClass {
  public static void main(String[] args) {
    double myDouble = 9.78;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

Run example »

Casting

- Casting is what a programmer does to explicitly convert a value from one type to another.
- The general syntax for a cast is:

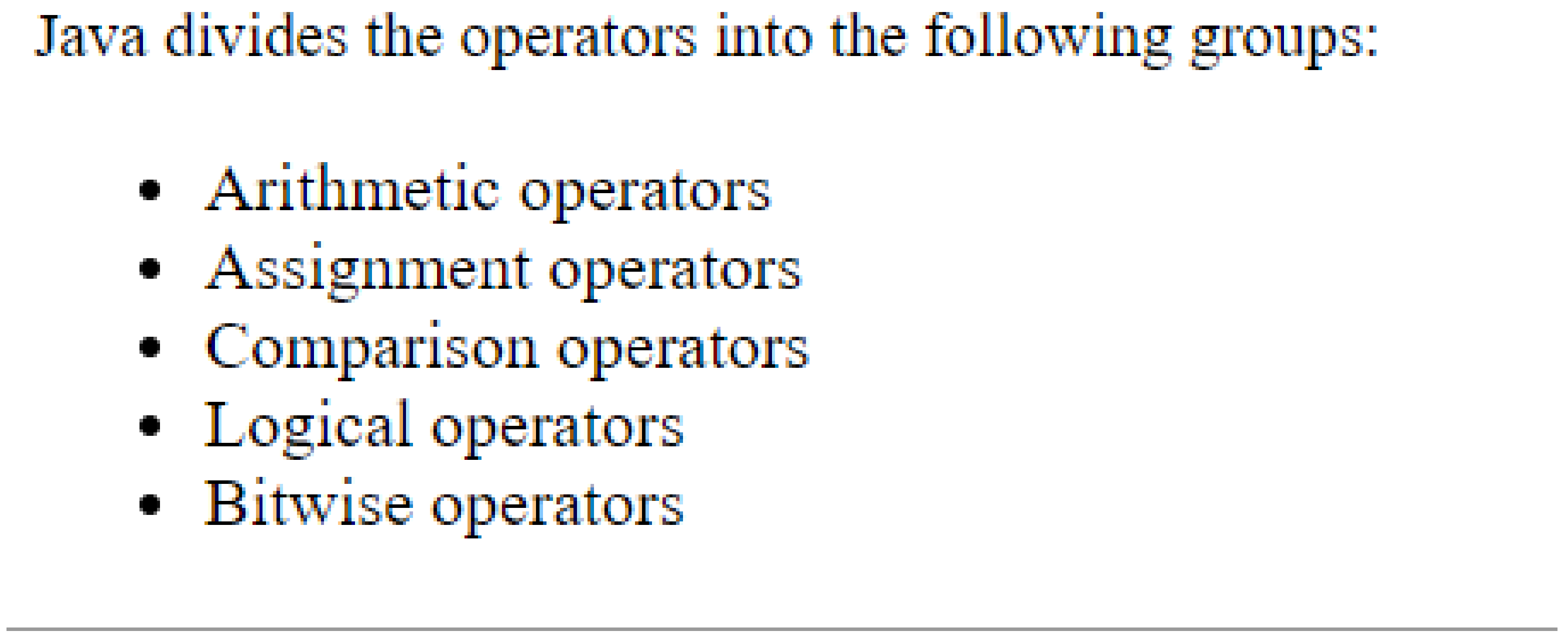
```
(result_type) value;
```

- Examples

```
float price = 37.53;
int dollars = (int) price;        // fractional portion lost
                        // dollars = 37

char response = 'A';
byte temp =(byte) response;  // temp = 65 (ASCII value
                        // for 'a')
```

Java Operators



Java divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value from another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Postfix versus Prefix notation

- When prefix notation is used, the operation is performed first and the result is evaluated.
- When postfix notation is used, the result is evaluated first and then the operation is performed.

```
int x = 5;          // x initialized to 5
int y = x++;        // y assigned the value of x, x incremented
               // ie x = 6, y = 5

int z = ++x;        // x incremented, z assigned the value of x
               // ie x = 7, z = 7
```

- Often used when accessing array indices:

```
int[] grades = {96, 74, 88, 56};
int index = 0;

     int firstGrade = grades[x++];
```

Assignment -Revisited

- Java also defines assignment operators which have an implied mathematical function

```
x = x + 1;              x += 1;
x = x + y + 5;          x += y + 5;
x = x * (z * 50);       x *= z * 50;
x = x / 10;             x /= 10;
```

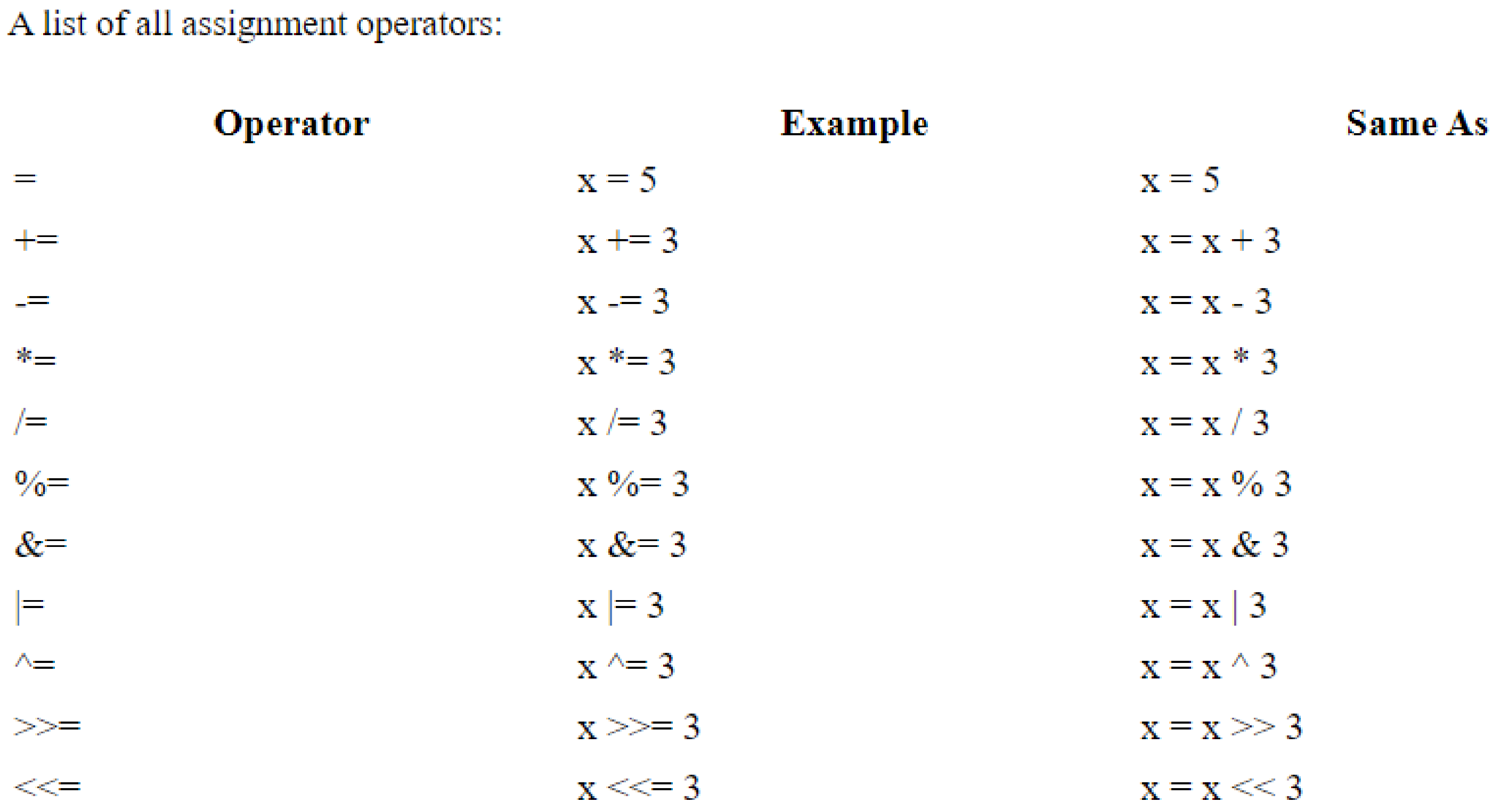
- These were originally added to the C language so that the programmer could help the compiler optimise expressions. It is generally better to avoid using these assignment operators.

- Be aware of the precedence issues:

```
x *= y + 5;    does not equal          x = x * y + 5;
           instead, it equals      x = x * (y + 5);
```

- Assignment ALWAYS has the lowest precedence.

Java Assignment Operators



A list of all assignment operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Java Comparison Operators

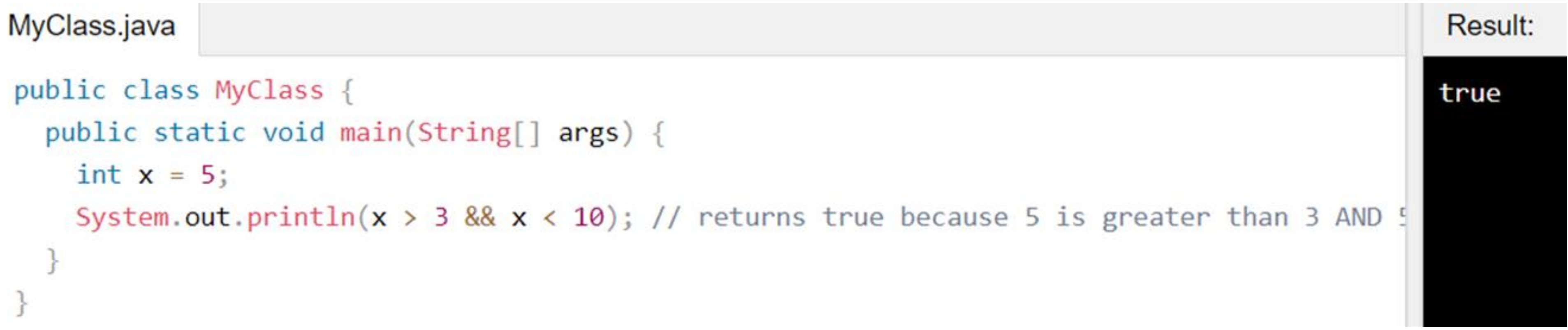
Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Java Logical Operators

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)



MyClass.java

```java
public class MyClass {
  public static void main(String[] args) {
    int x = 5;
    System.out.println(x > 3 && x < 10); // returns true because 5 is greater than 3 AND 5
  }
}
```

Result:

```
true
```

Java Strings

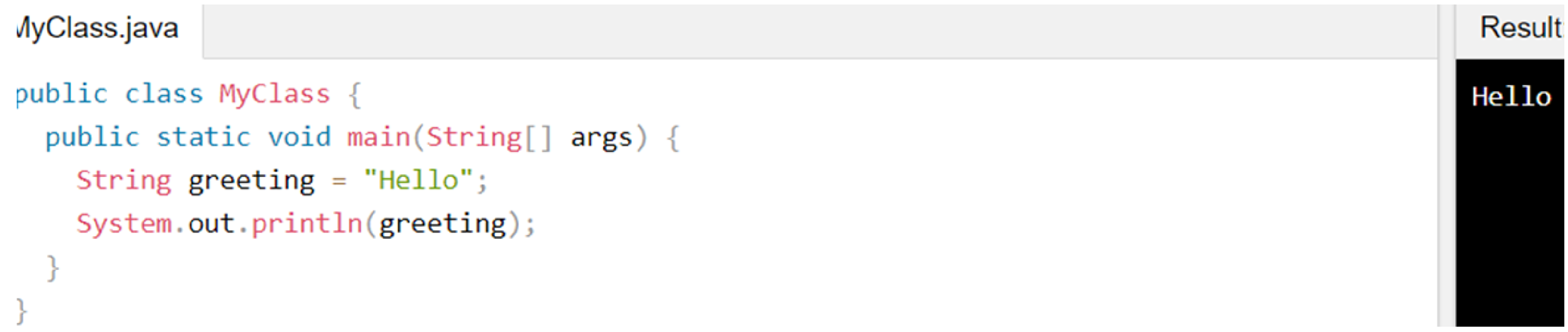
Strings are used for storing text.

A `String` variable contains a collection of characters surrounded by double quotes:

Example

Create a variable of type `String` and assign it a value:

```java
String greeting = "Hello";
```



```java
public class MyClass {
  public static void main(String[] args) {
    String greeting = "Hello";
    System.out.println(greeting);
  }
}
```

```
Hello
```

More String Methods

There are many string methods available, for example `toUpperCase()` and `toLowerCase()`:

Example

```
String txt = "Hello World";
System.out.println(txt.toUpperCase());   // Outputs "HELLO WORLD"
System.out.println(txt.toLowerCase());   // Outputs "hello world"
```

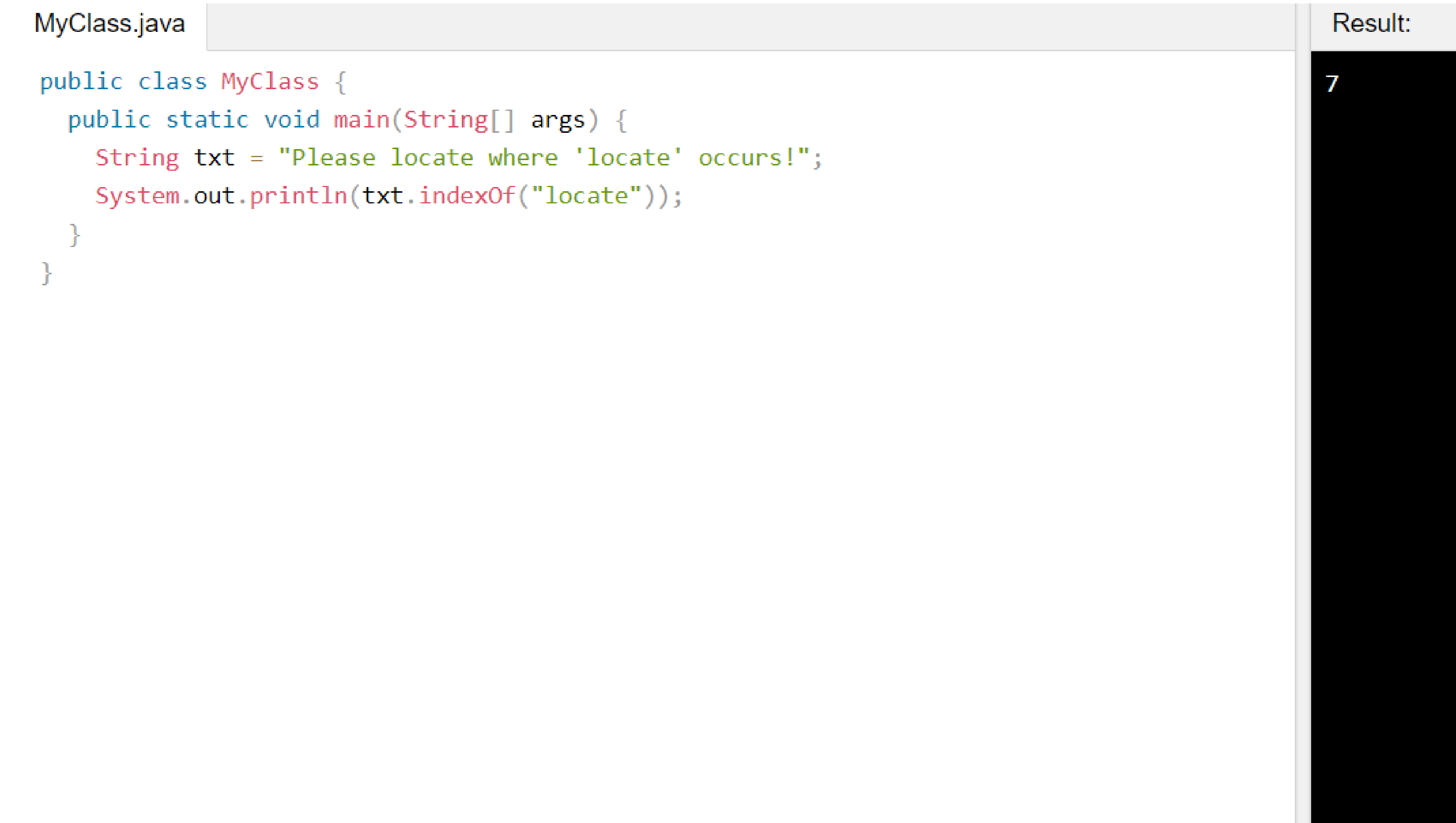
Run example »

Example

```
String txt = "Please locate where 'locate' occurs!";
System.out.println(txt.indexOf("locate")); // Outputs 7
```

Run example »

Example



MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    String txt = "Please locate where 'locate' occurs!";
    System.out.println(txt.indexOf("locate"));
  }
}
```

Result:

```
7
```

String Concatenation

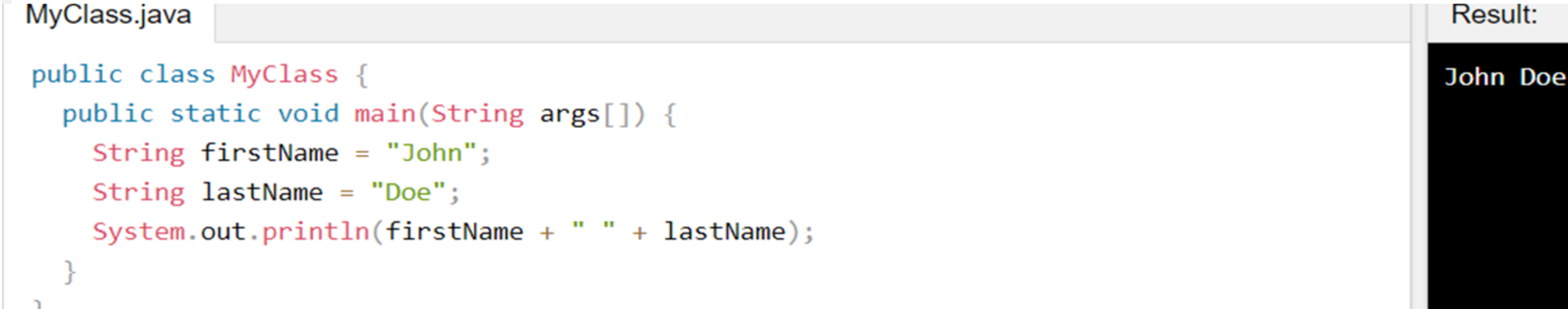
The **+** operator can be used between strings to combine them. This is called **concatenation**:

Example

```
String firstName = "John";
String lastName = "Doe";
System.out.println(firstName + " " + lastName);
```

Run example »

Note that we have added an empty text (" ") to create a space between firstName and lastName on print.



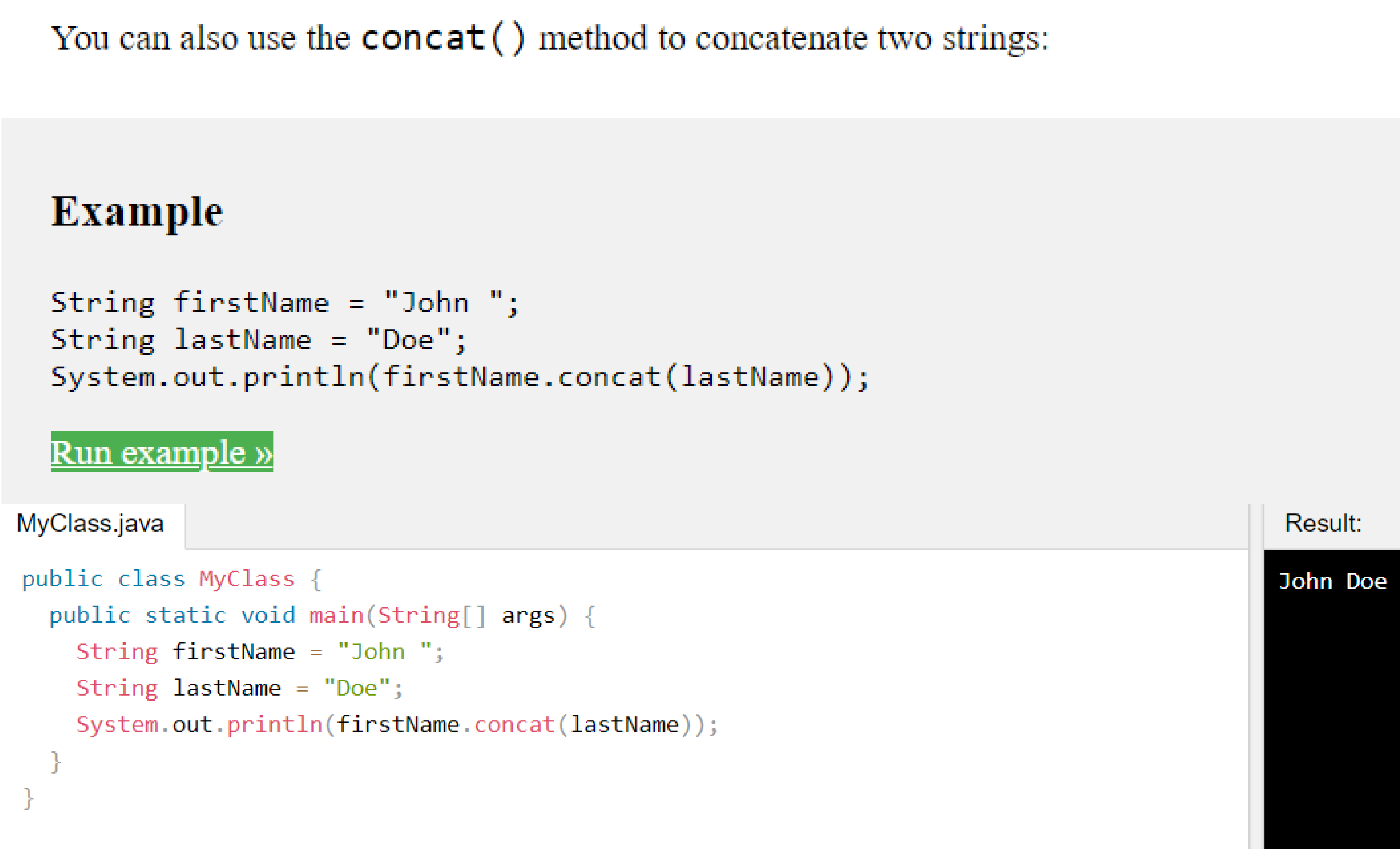
MyClass.java

```
public class MyClass {
  public static void main(String args[]) {
    String firstName = "John";
    String lastName = "Doe";
    System.out.println(firstName + " " + lastName);
  }
}
```

Result:

```
John Doe
```

Example



You can also use the `concat()` method to concatenate two strings:

Example

```
String firstName = "John ";
String lastName = "Doe";
System.out.println(firstName.concat(lastName));
```

Run example »

MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    String firstName = "John ";
    String lastName = "Doe";
    System.out.println(firstName.concat(lastName));
  }
}
```

Result:

```
John Doe
```

Special Characters

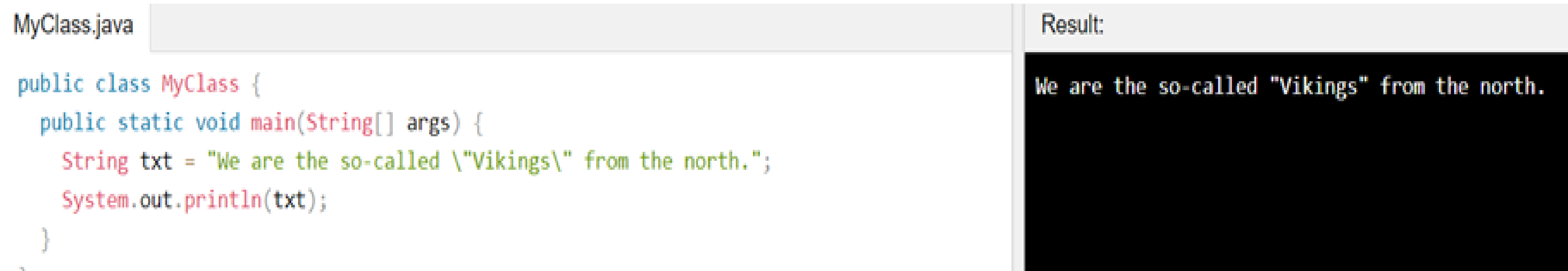
```
String txt = "We are the so-called "Vikings" from the north.";
```

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

The sequence \" inserts a double quote in a string:



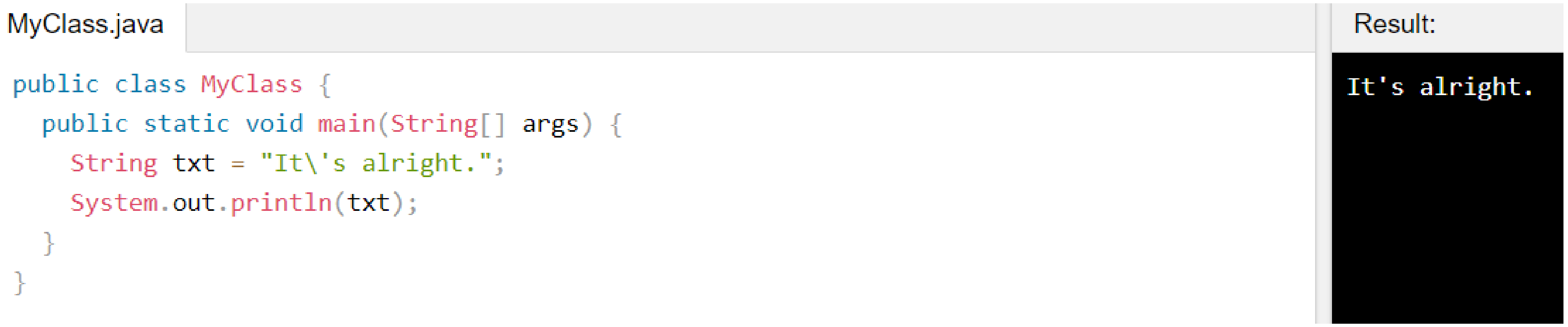
MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    String txt = "We are the so-called \"Vikings\" from the north.";
    System.out.println(txt);
  }
```

Result:

```
We are the so-called "Vikings" from the north.
```

Examples

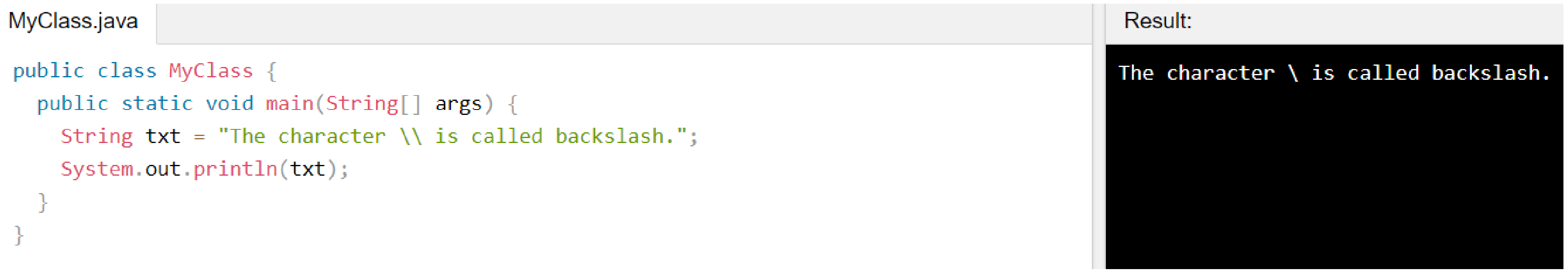


MyClass.java

```java
public class MyClass {
  public static void main(String[] args) {
    String txt = "It\'s alright.";
    System.out.println(txt);
  }
}
```

Result:

```
It's alright.
```



MyClass.java

```java
public class MyClass {
  public static void main(String[] args) {
    String txt = "The character \\ is called backslash.";
    System.out.println(txt);
  }
}
```

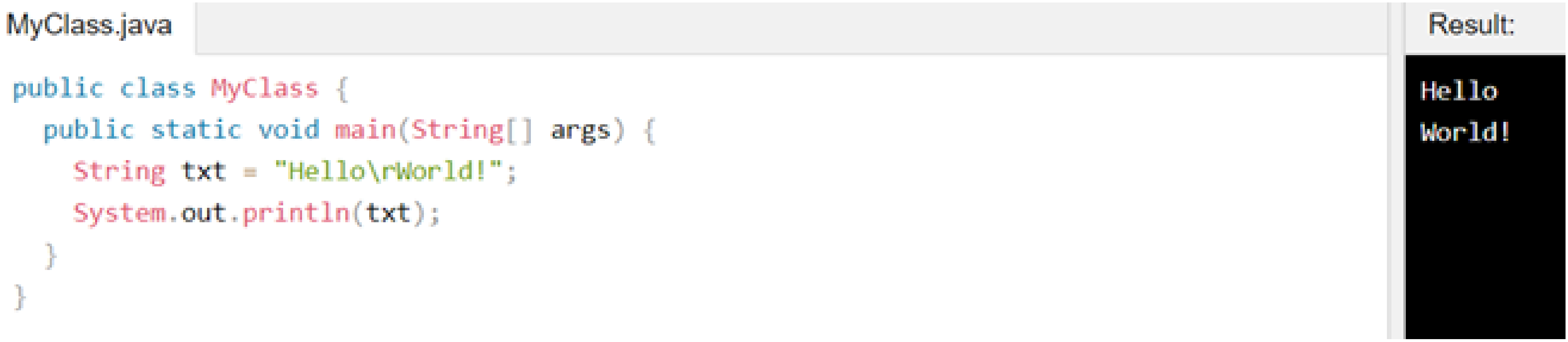
Result:

```
The character \ is called backslash.
```

Special Characters

Six other escape sequences are valid in Java:

Code	Result
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed



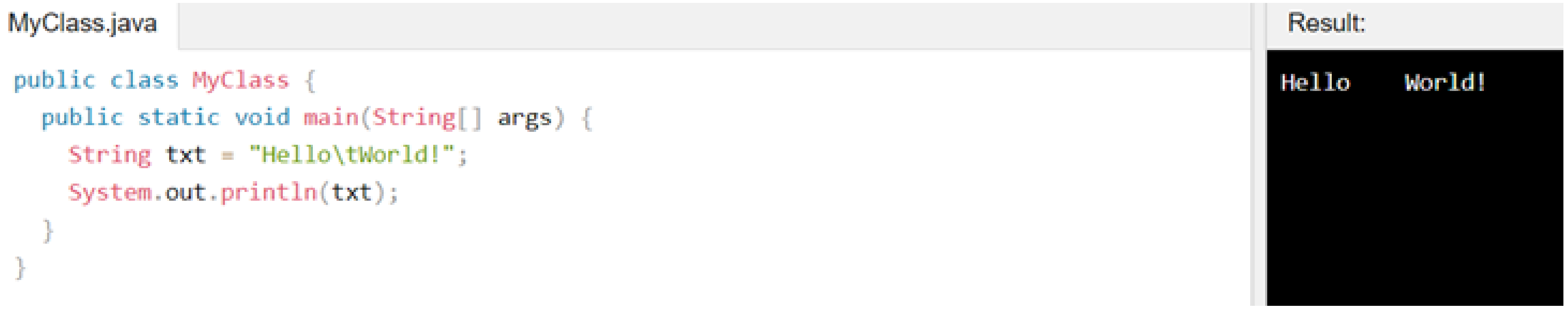
MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    String txt = "Hello\rWorld!";
    System.out.println(txt);
  }
}
```

Result:

```
Hello
World!
```

Examples

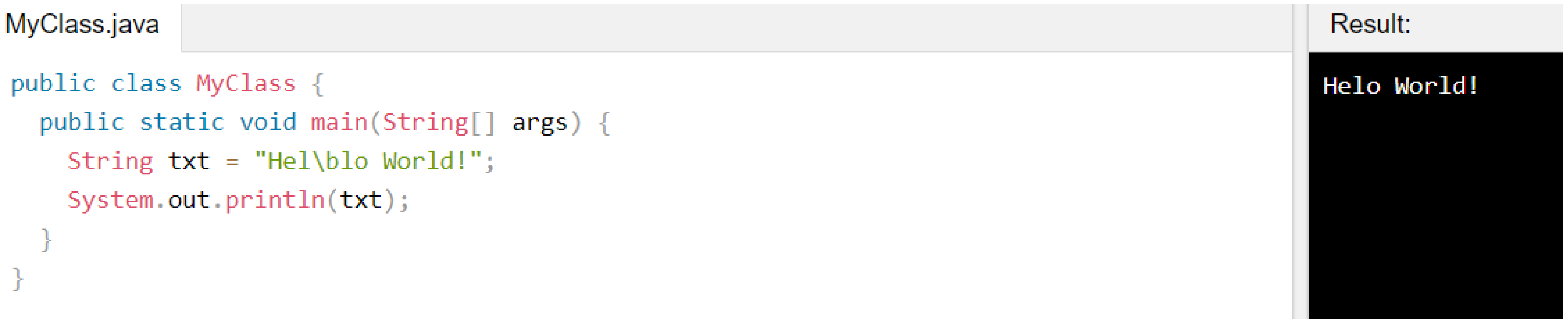


MyClass.java

```java
public class MyClass {
  public static void main(String[] args) {
    String txt = "Hello\tWorld!";
    System.out.println(txt);
  }
}
```

Result:

```
Hello     World!
```



MyClass.java

```java
public class MyClass {
  public static void main(String[] args) {
    String txt = "Hel\blo World!";
    System.out.println(txt);
  }
}
```

Result:

```
Helo World!
```

Reserved Words in Java

- boolean
- byte
- char
- short
- int
- long
- float
- double
- void

- false
- null
- true

- abstract
- final
- native
- private
- protected
- public
- static
- synchronized
- transient
- volatile

- break
- case
- catch
- continue
- default
- do
- else
- finally
- for
- if
- return
- switch
- throw
- try
- while

- class
- extends
- implements
- interface
- throws

- import
- package

- instanceof
- new
- super
- this

- byvalue
- cast
- const
- future
- generic
- goto
- inner
- operator
- outer
- rest
- var

reserved for future use.

Tips for good variable names

- Use a naming convention
- Use names which are meaningful within their context
- Start Class names with an Upper case letter. Variables and other identifiers should start with a lower case letter.
- Avoid using _ and $.
- Avoid prefixing variable names (eg. _myAge, btnOk)
  - This is often done in languages where type is not strongly enforced.
  - If you do this in Java, it is often an indication that you have not chosen names meaningful within their context.
- Separate words with a capital letter, not an underscore (_)
  - myAccount, okButton, aLongVariableName
  - avoid: my_account, ok_button, and a_long_variable_name

Initializing variables

- Although Java provides ALL variables with an initial value, variables should be initialized before being used.
- Java allows for initializing variables upon declaration.
- It is considered good practice to initialize variables upon declaration.
- Variables declared within a method must be initialized before use or the compiler with issue an error.

```
int total = 100;
float xValue = 0.0;
boolean isFinished = false;
String name = "Zippy The Pinhead";
```

Constant Values

- A variable can be made constant by including the keyword *final* in its declaration.
- By convention, the names of variables defined as final are UPPER CASE.
- Constants allow for more readable code and reduced maintenance costs.
- Final variables must be initialized upon declaration.

```
final int MAX_BUFFER_SIZE
= 256;
final float PI=3.14159;
```