Unit 1

Basics of Java Programming Language

Contents

Introduction to Java, Features, Java Programming Environment and Java Virtual Machine, Program structure, Naming conventions, Data types, Operators, Reading and displaying input/output, Type Casting, Garbage Collection, Command Line Arguments, Variable length Arguments, finalize() method

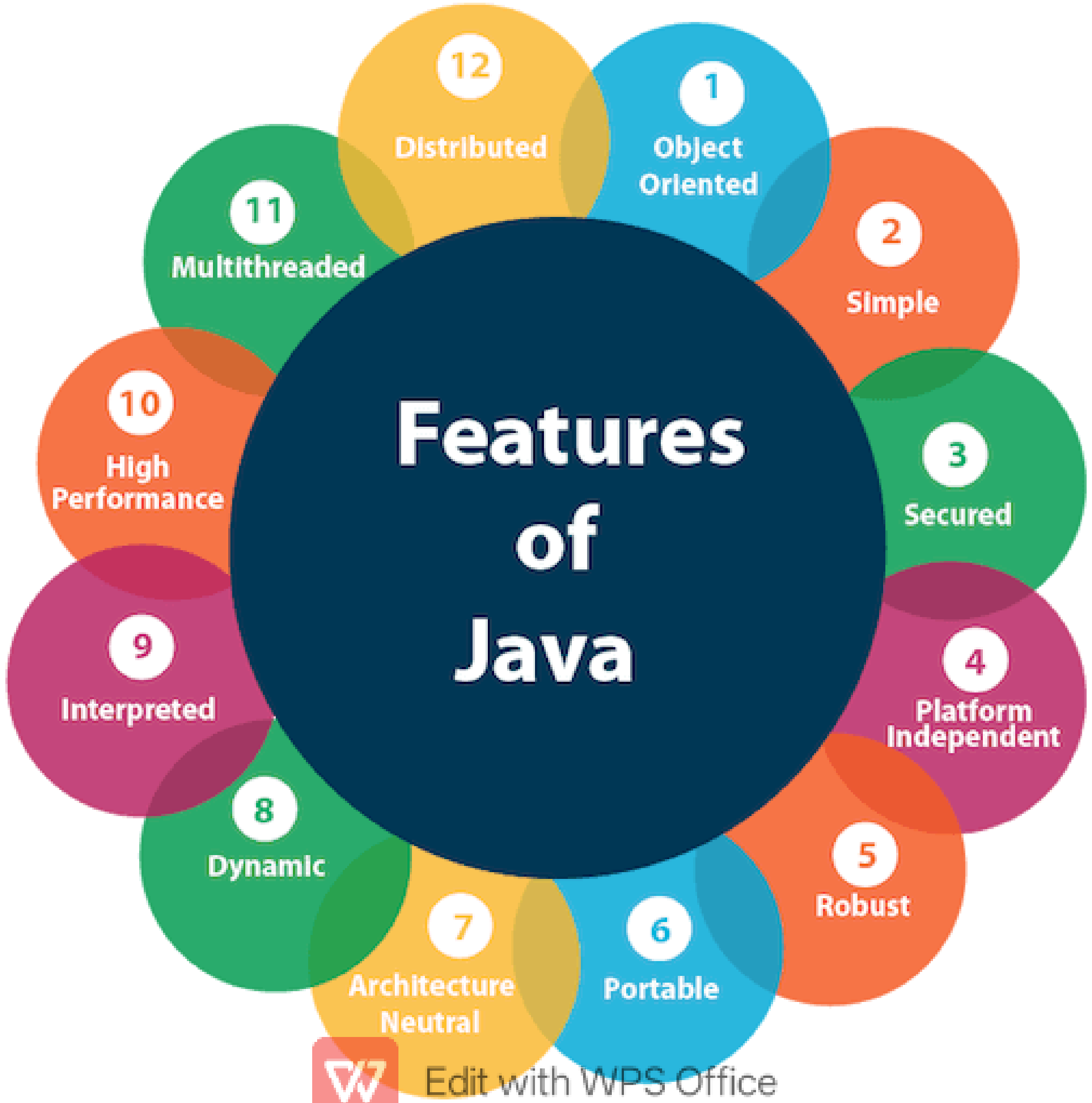
Access Specifiers: public, private, protected, default,

Strings: String class and its functions, Wrapper classes.

Case Study: *Demonstrate Online Bookstore Management System for a local bookstore using basics concepts of java.*

Introduction to Java

- Developed by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991
- It took 18 months to develop the first working version.
- It was initially called “Oak”, but was renamed “Java” in 1995.
- Java Versions - 1.0, 1.1, 2, J2SE (Java 2 Platform Standard Edition), J2SE 1.3, 1.4, J2SE 5, 6, 7, 8, 9, 10, 11
- Object-oriented programming (OOP) is at the core of Java
- Java derives much of its character from C and C++


Features of Java
1 Object Oriented
2 Simple
3 Secured
4 Platform Independent
5 Robust
6 Portable
7 Architecture Neutral
8 Dynamic
9 Interpreted
10 High Performance
11 Multithreaded
12 Distributed

Features of Java

Simple

- Java is very easy to learn, syntax is simple, clean and easy to understand
- Java syntax is based on C++.
- Java has removed many complicated and rarely-used features, like, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

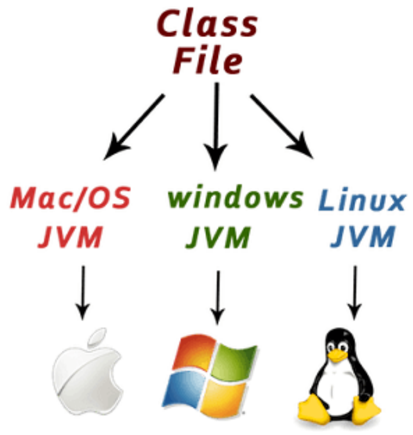
Object-oriented

- Java is an object-oriented programming language.
- Everything in Java is an object.
- Basic OOPs concepts are: Object, Class, Inheritance, Polymorphism, Abstraction, Encapsulation

Features of Java

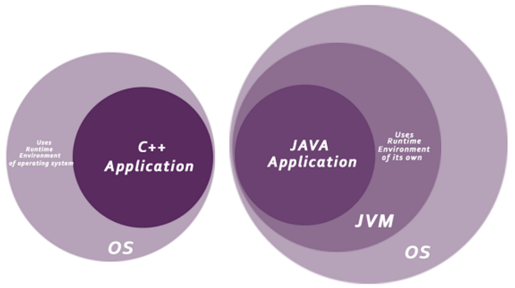
Platform Independent

- Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc.
- Java code is compiled by the compiler and converted into bytecode.
- This bytecode is platform-independent because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).



Secured

- Java is best known for its security.
- With Java, we can develop virus-free systems
- Java is secured because:
  - No explicit pointer
  - Run inside a virtual machine sandbox


Uses Runtime Environment of operating system
C++ Application
OS
JAVA Application
Uses Runtime Environment of its own
JVM
OS

Features of Java

Multi-threaded

- A thread is like a separate program, executing concurrently.
- We can write Java programs that deal with many tasks at once by defining multiple threads.
- The main advantage of multi-threading is that it doesn't occupy memory for each thread.
- It shares a common memory area. Threads are important for multimedia, Web applications, etc.

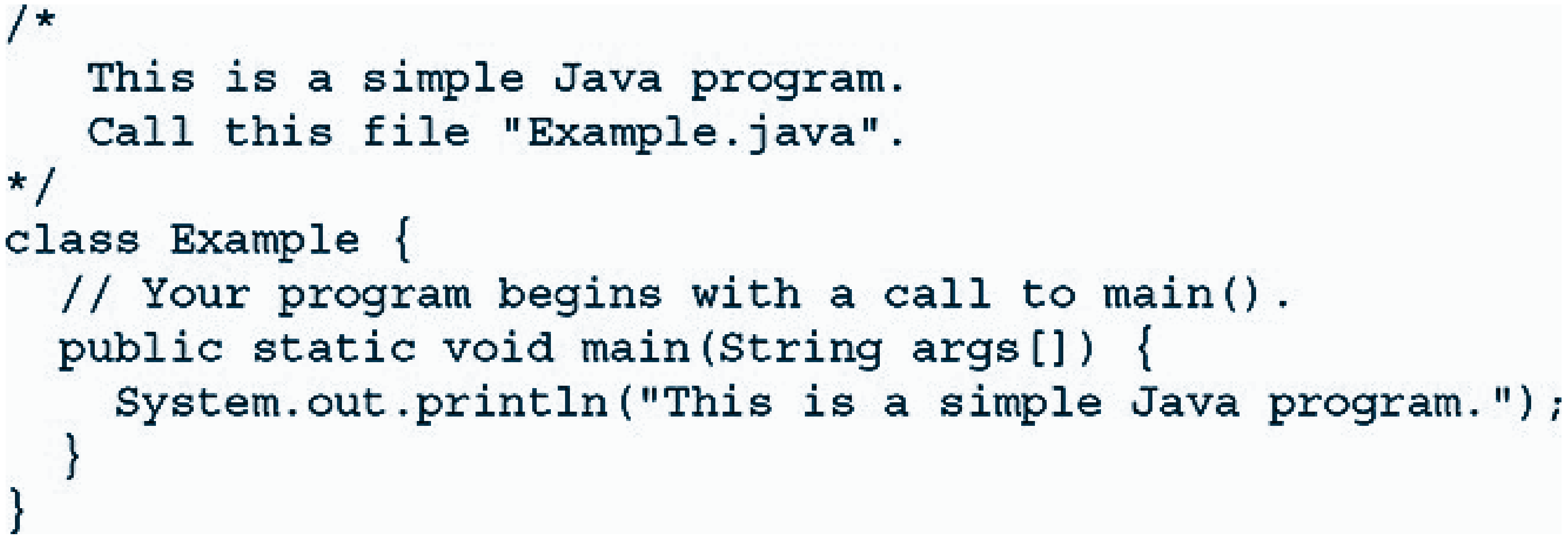
Dynamic

- Java is a dynamic language.
- It supports the dynamic loading of classes. It means classes are loaded on demand.
- It also supports functions from its native languages, i.e., C and C++.

Java Programming Environment and JVM

- Output of a Java compiler is not executable code, it is bytecode
- Bytecode is a highly optimized set of instructions designed to be executed by Java Virtual Machine (JVM)
- This JVM is the part of Java Runtime Environment (JRE)
- Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments
- Once a JRE exists for the given system, any Java program can run on it with the help of JVM

Program structure



```
/*
   This is a simple Java program.
   Call this file "Example.java".
*/
class Example {
  // Your program begins with a call to main().
  public static void main(String args[]) {
    System.out.println("This is a simple Java program.");
  }
}
```

Program structure

- For this example, name of source file should be ***Example.java***
- To compile the program, execute the compiler, ***javac***, specifying the name of the source file on the command line,

  C:\>java\bin\javac Example.java
- To actually run the program, you must use the Java application launcher called java. To do so, pass the class name Example as a command-line argument, as shown here:

  C:\>java\bin\java Example

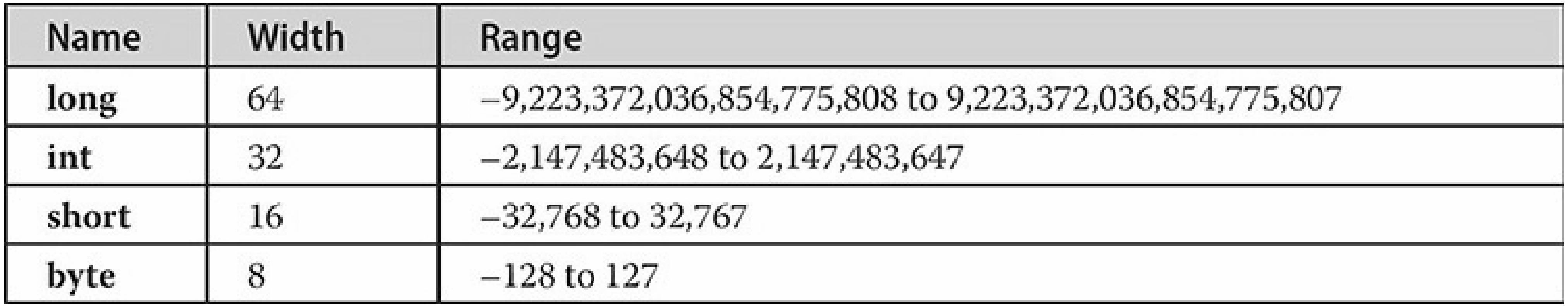
Data types

- Java defines eight primitive types of data: *byte, short, int, long, char, float, double, and boolean*

- These can be put in four groups:
  - **Integers -** This group includes byte, short, int, and long, which are for whole-valued signed numbers.
  - **Floating-point numbers -** This group includes float and double, which represent numbers with fractional precision.
  - **Characters -** This group includes char, which represents symbols in a character set, like letters and numbers.
  - **Boolean -** This group includes boolean, which is a special type for representing true/false values.

Integer

- Integers - Java defines four integer types: *byte, short, int & long*
- All of these are signed, positive and negative values



Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Byte

- The smallest integer type is byte. This is a signed 8-bit type that has a range from –128 to 127.
- Variables of type byte are especially useful when you’re working with a stream of data from a network or file.
- Byte variables are declared by use of the *byte* keyword

Short

- *short* is a signed 16-bit type.
- It has a range from –32,768 to 32,767
- It is probably the least-used Java type

Int

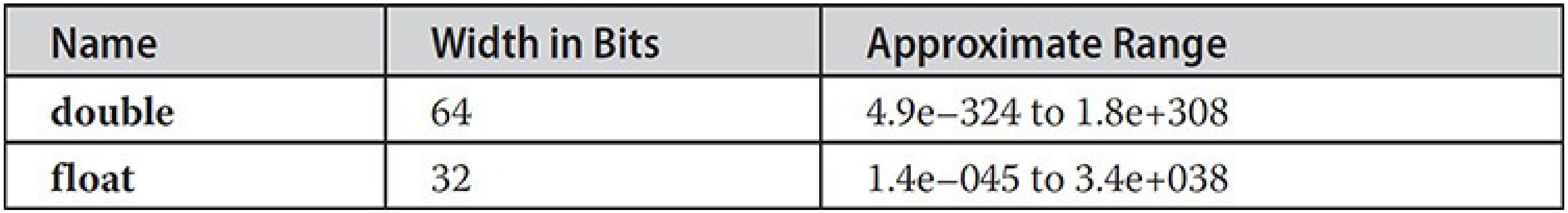
- The most commonly used integer type is *int*
- It is a signed 32-bit type that has a range from –2,147,483,648 to 2,147,483,647.
- In addition to other uses, variables of type int are commonly employed to control loops and to index arrays.

Long

- Signed 64-bit type and
- Useful for those occasions where an int type is not large enough to hold the desired value
- The range of a long is quite large
- This makes it useful when big, whole numbers are needed

Floating-point Numbers

- Also known as *real numbers*
- Used when evaluating expressions that require fractional precision
- There are two kinds of floating point types, float and double



Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

Float

- Specifies a single-precision value that uses 32 bits of storage
- Single precision is faster on some processors and takes half as much space as double precision
- Variables of type float are useful when you need a fractional component, but don’t require a large degree of precision
- For example, float can be useful when representing dollars and cents

Double

- Denoted by the *double* keyword uses 64 bits to store a value
- Double precision is actually faster than single precision on some modern processors
- All transcendental math functions, such as *sin( )*, *cos( )*, and *sqrt( )*, return double values
- When you need to maintain accuracy over many iterative calculations, or are manipulating large-valued numbers, double is the best choice

Characters

- In Java, the data type used to store characters is char
- Java uses Unicode to represent characters
- Unicode defines a fully international character set that can represent all of the characters found in all human languages
- It is unification of dozens of character sets, like Latin, Greek, Arabic, Cyrillic, Hebrew, Katakana, Hangul, and many more
- At the time of Java's creation, Unicode required 16 bits. Thus, in Java char is a 16-bit type

Characters

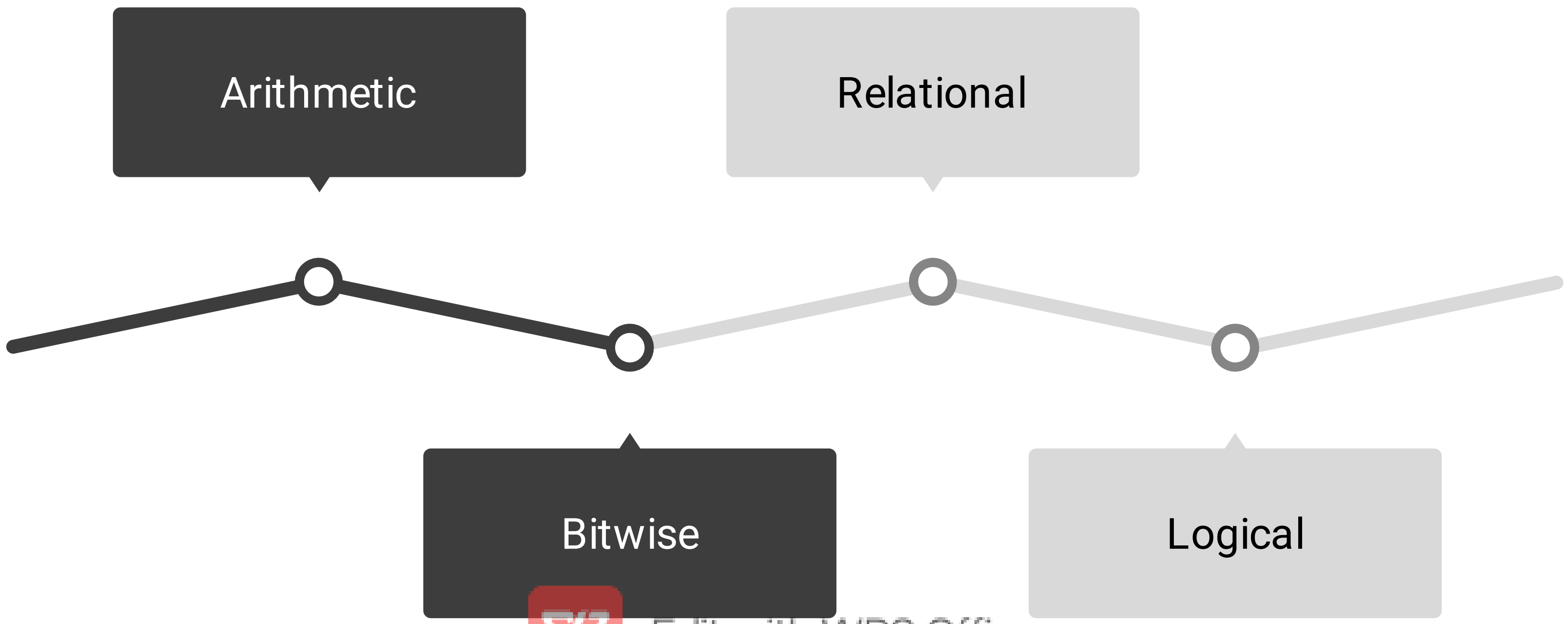
- The range of a char is 0 to 65,536. There are no negative chars.
- The standard set of characters known as ASCII still ranges from 0 to 127 as always
- And the extended 8-bit character set, ISO-Latin-1, ranges from 0 to 255
- Although char is designed to hold Unicode characters, it can also be used as an integer type on which you can perform arithmetic operations

Booleans

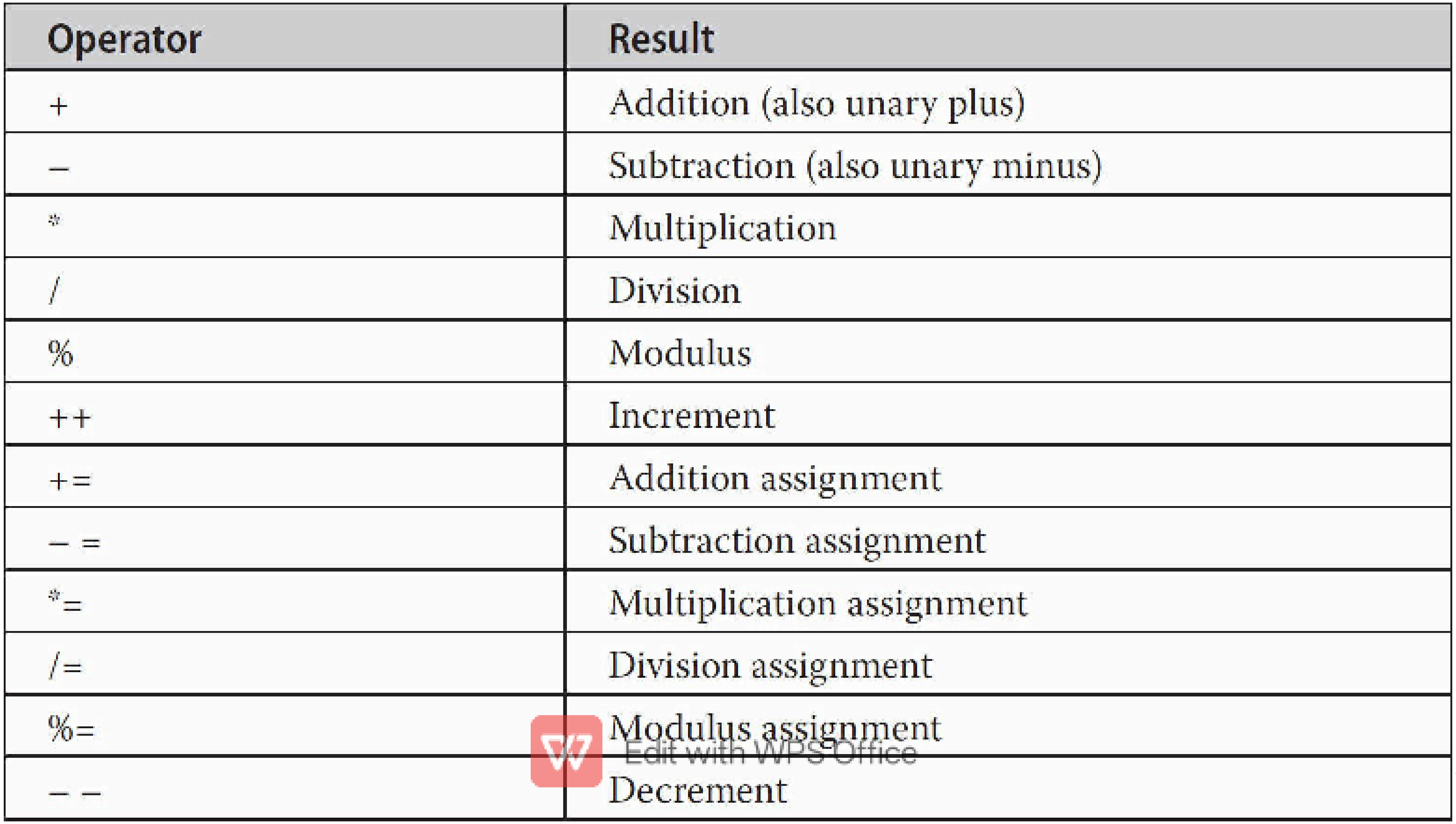
- Java has a primitive type, called *boolean*, for logical values
- It can have only one of two possible values, *true* or *false*
- This is the type returned by all *relational* operators, as in the case of a < b.
- Boolean is also the type required by the *conditional* expressions that govern the control statements such as *if* and *for*

Operators

- Java provides a rich operator environment.
- Most of them can be divided into the following four groups



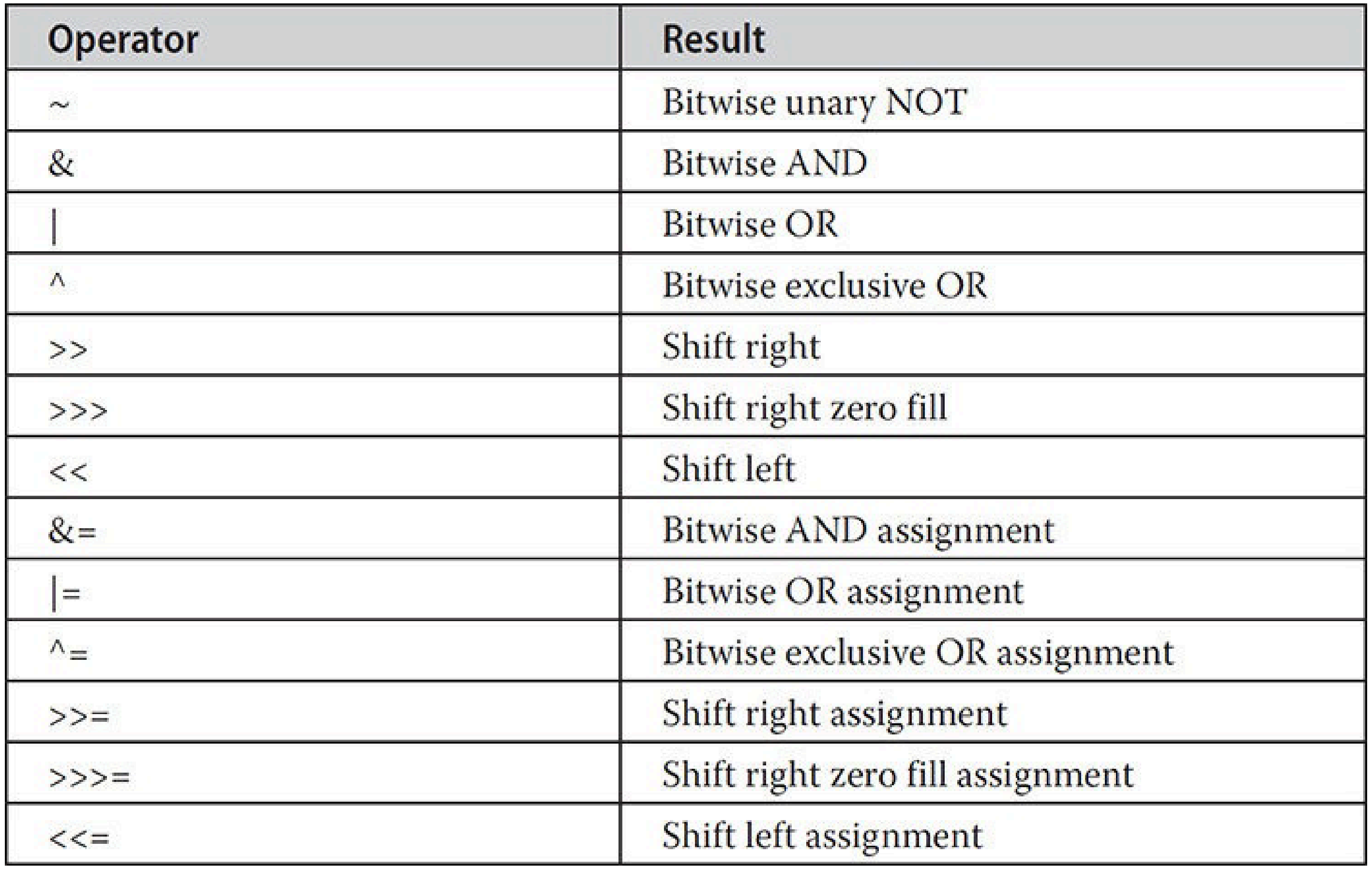
Arithmetic Operators



Operator	Result
+	Addition (also unary plus)
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
– =	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

Bitwise Operators

- Java defines several bitwise operators that can be applied to the integer types: long, int, short, char, and byte



Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Relational Operators

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

The outcome of these operations is a *boolean* value

Logical Operators

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

Reading Input

- Use Scanner for simple console input
- Use FileReader/FileWriter for small file reading/writing
- Use BufferedReader/BufferedWriter for large files
- Use FileInputStream/FileOutputStream for binary data (images, videos)
- Use FileReader/FileWriter for text data

Reading Input using Scanner

- The Scanner class in Java is part of the java.util package
- It is used to read input from various sources such as the keyboard, files, and input streams.
- To use the Scanner class, you must first import it:

1. Importing the Scanner Class

```java
import java.util.Scanner;
```

2. Creating a Scanner Object

```java
Scanner scanner = new Scanner(System.in);
```

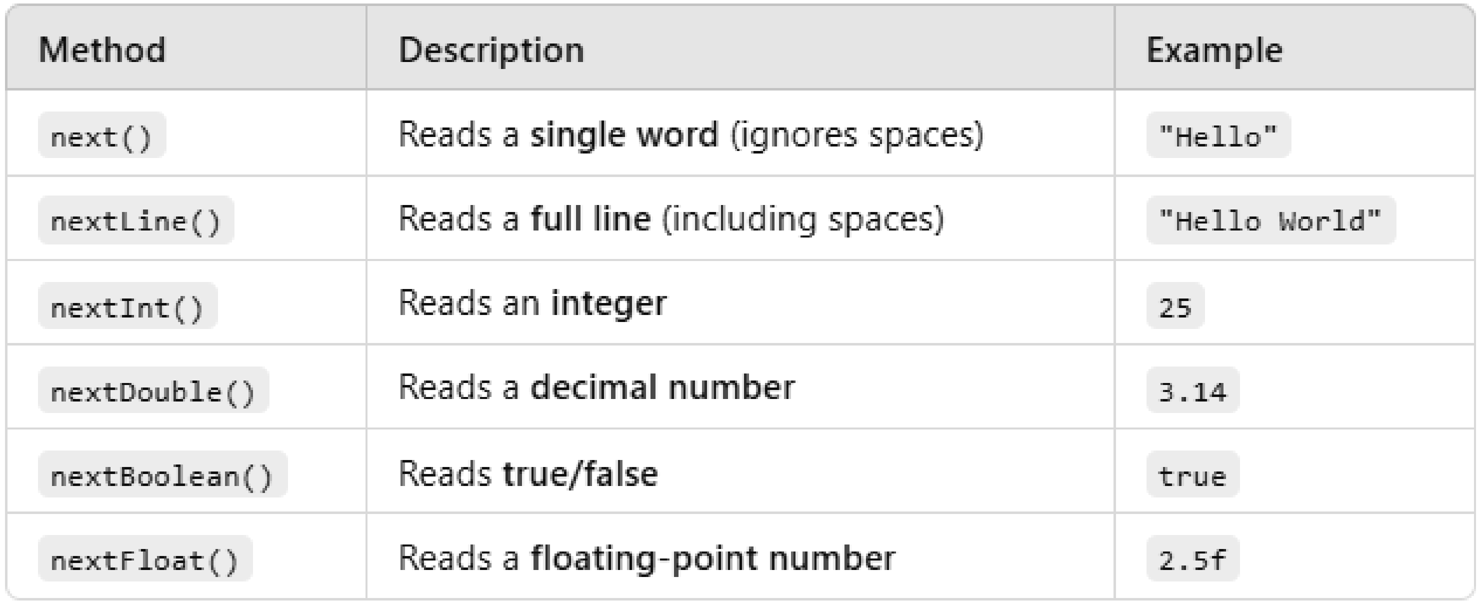
Scanner → Class name

scanner → Object of Scanner class

System.in → Takes input from the keyboard

Reading Input using Scanner

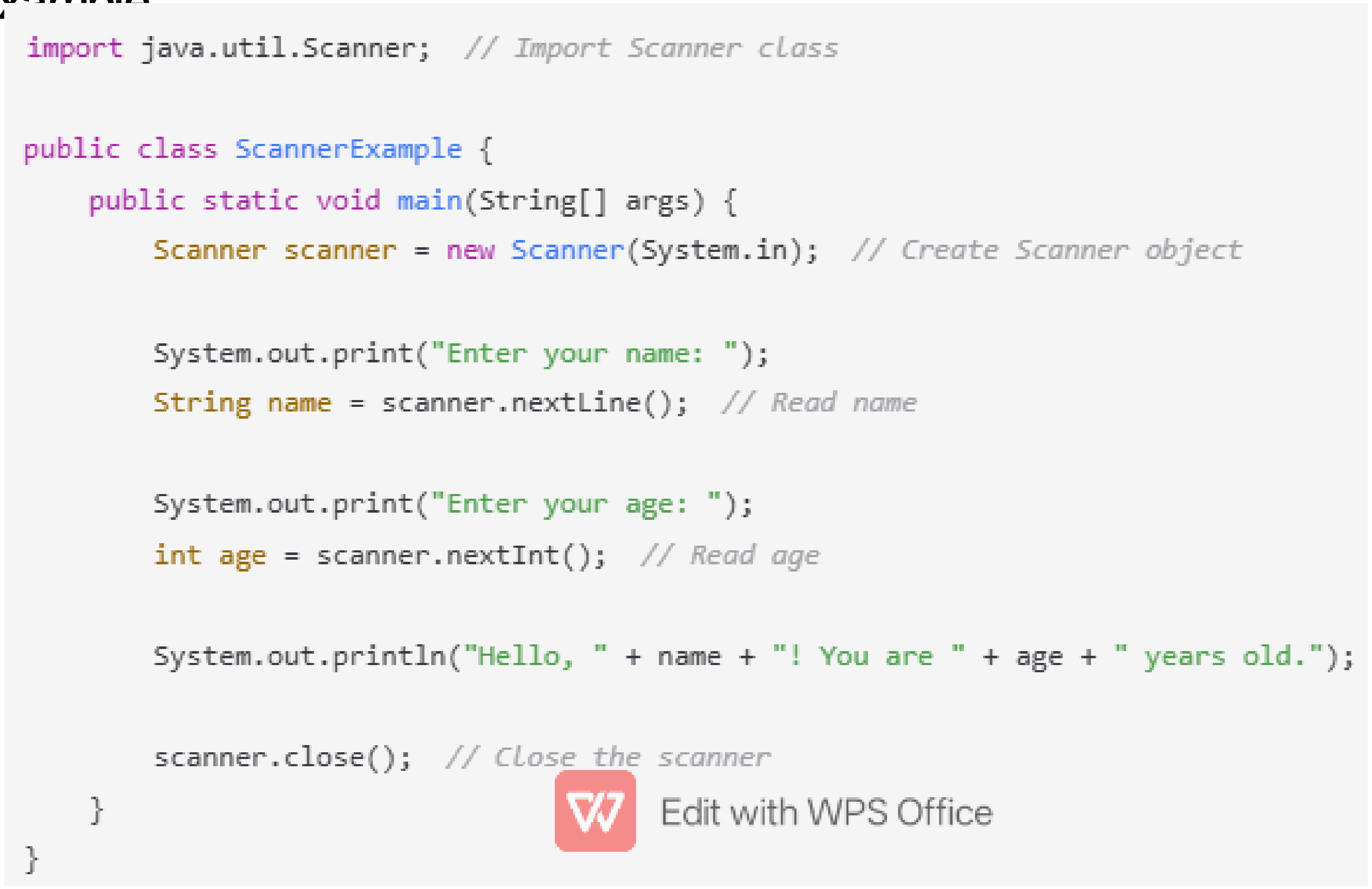
3. Taking Different Types of Input



Method	Description	Example
`next()`	Reads a **single word** (ignores spaces)	`"Hello"`
`nextLine()`	Reads a **full line** (including spaces)	`"Hello World"`
`nextInt()`	Reads an **integer**	`25`
`nextDouble()`	Reads a **decimal number**	`3.14`
`nextBoolean()`	Reads **true/false**	`true`
`nextFloat()`	Reads a **floating-point number**	`2.5f`

Reading Input using Scanner

Example:



```java
import java.util.Scanner;  // Import Scanner class

public class ScannerExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);  // Create Scanner object

        System.out.print("Enter your name: ");
        String name = scanner.nextLine();  // Read name

        System.out.print("Enter your age: ");
        int age = scanner.nextInt();  // Read age

        System.out.println("Hello, " + name + "! You are " + age + " years old.");

        scanner.close();  // Close the scanner
    }
}
```

Reading Input using Scanner

Output Example:

```
javac ScannerExample.java
```

```
java ScannerExample
```

Enter Input & Get Output:

```
Enter your name: Alice
Enter your age: 25
Hello, Alice! You are 25 years old.
```

Reading Input

- In Java, console input is taken by using the combination of System.in, BufferedReader, InputStreamReader
- Syntax is,

```
InputStreamReader isr = new InputStreamReader(System.in);

BufferedReader br = new BufferedReader(isr);
```

- ❑ Here, System.in - To read console input
- ❑ BufferedReader - To obtain character-based stream attached to console
- ❑ InputStreamReader - Converts bytes to characters

Reading Input

- To read a character from a BufferedReader, use *read( )*

  Syntax - int read( ) throws IOException

  Example: char c = (char) br.read()

- To read a string from the keyboard, use *readLine( )*

  Syntax - String readLine( ) throws IOException

  Example: String str = br.readLine()

Displaying Output

- Console output is most easily executed by *print( )* and *println( )*
- These methods are defined by the class PrintStream (which is the type of object referenced by System.out)
- Because PrintStream is an output stream derived from OutputStream, it also implements the low-level method write( )
- Thus, write( ) can be used to write to the console

void write(int byteval)

Displaying Output

- Although using System.out to write to console is acceptable, recommended method of writing to console is using *PrintWriter*
- PrintWriter is one of the character-based classes,

PrintWriter(OutputStream outputStream, boolean flushingOn)

- ❑ Here, outputStream - object of type OutputStream
- ❑ flushingOn - controls whether Java flushes output stream every time a println( ) method is called.
- ❑ If flushingOn is true, flushing automatically takes place. Else it is not automatic.

Displaying Output

- PrintWriter supports the *print( )* and *println( )* methods
- To write to the console by using a *PrintWriter*, specify *System.out* for the output stream and automatic flushing
- Example,

```
PrintWriter pw = new PrintWriter(System.out, true);
pw.println("Hello World!");
```

Type Casting

- It is common to assign a value of one type to a variable of another type
- If the two types are compatible, then Java will perform the conversion automatically.
- Example, it’s always possible to assign *int* value to *long* variable
- But no automatic conversion defined from *double* to *byte*
- Fortunately, it is still possible to obtain a conversion between incompatible types using *type casting*

Type Casting

- Although the automatic type conversions are helpful, they will not fulfill all needs
- If you want to assign an *int* value to a *byte* variable then this conversion will not be performed automatically, because a *byte* is smaller than an *int*
- To create a conversion between two incompatible types, you must use a *cast*

Type Casting

- A *cast* is simply an explicit type conversion
- It’s general form - *(target-type) value*
- For example,

```
int a;
byte b;
// ...
b = (byte) a;
```

Garbage Collection

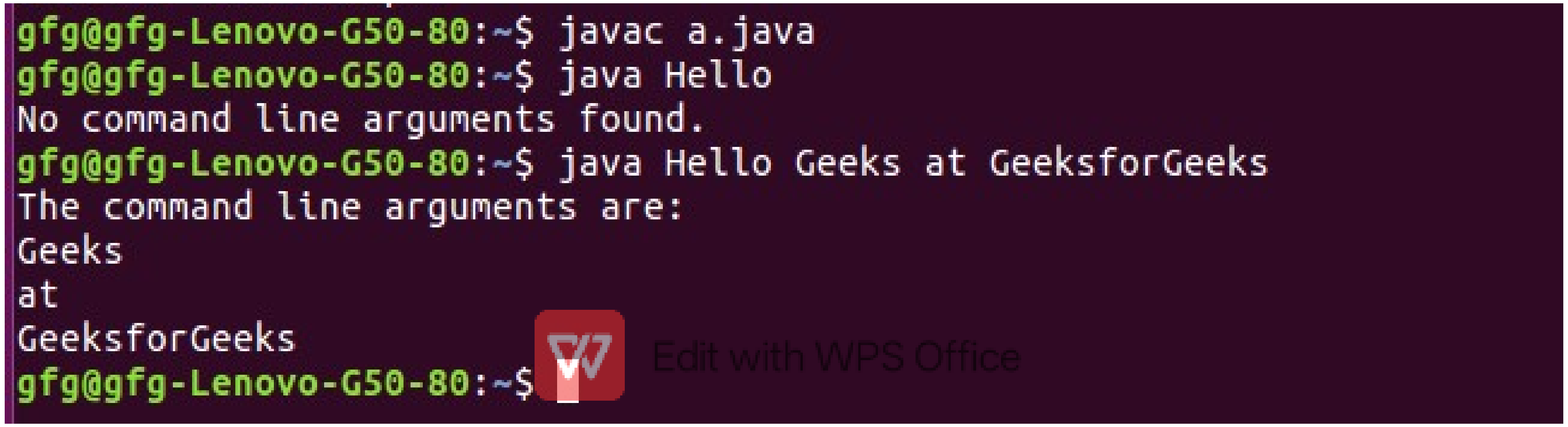
- Objects are dynamically allocated by using the new operator
- In some languages, like C++, dynamically allocated objects must be manually released by use of *delete* operator
- Java handles deallocation automatically
- The technique that manages this is called *garbage collection*
- If no reference to object exist, that object is assumed to be no longer needed, & memory occupied by object can be reclaimed
- There is no need to explicitly destroy objects

Command Line Arguments

- Sometimes you will want to pass information into a program when you run it
- This can be done by passing command-line arguments to *main*
- A command-line argument is the information that directly follows program's name on command line when it is executed
- To access command-line arguments in Java is quite easy
- They are stored as strings in a *String* array passed to the *args* parameter of *main( )*

Steps for Command Line Arguments

- Save the program as Hello.java
- Open command prompt and compile program- javac Hello.java
- After successful compilation of the program, run the following command by writing the arguments- java Hello
- For example – java Hello Geeks at GeeksforGeeks & press Enter

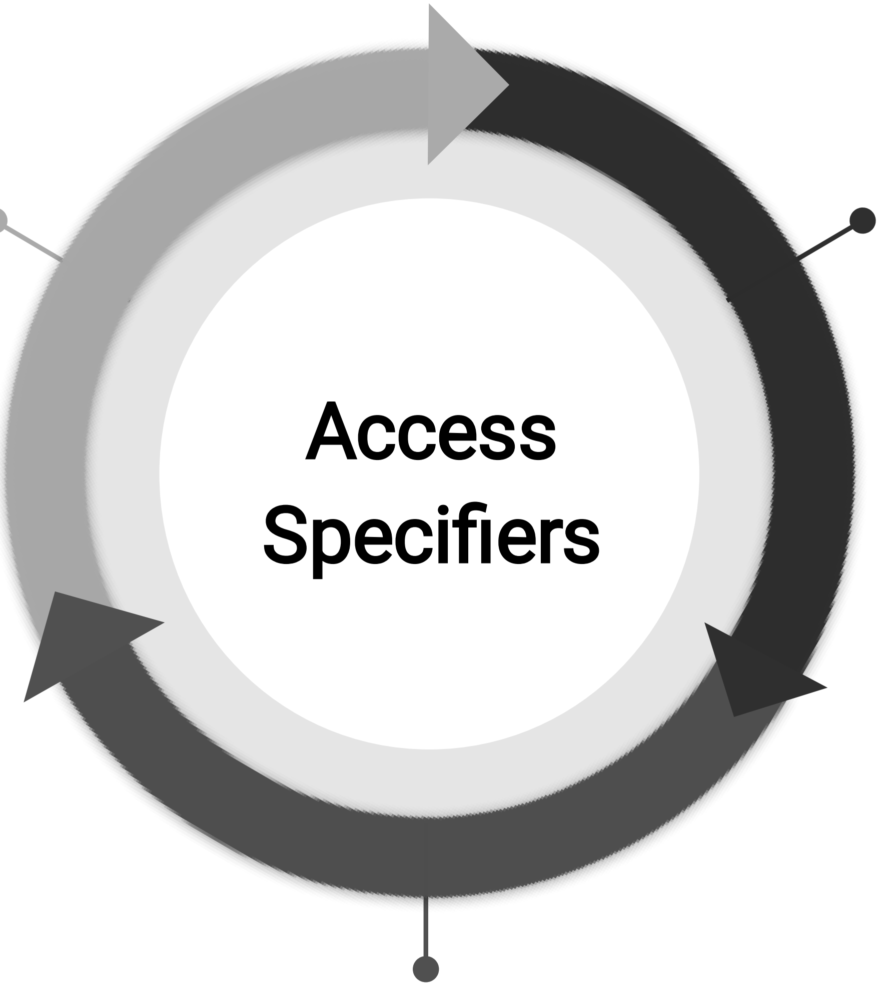


```
gfg@gfg-Lenovo-G50-80:~$ javac a.java
gfg@gfg-Lenovo-G50-80:~$ java Hello
No command line arguments found.
gfg@gfg-Lenovo-G50-80:~$ java Hello Geeks at GeeksforGeeks
The command line arguments are:
Geeks
at
GeeksforGeeks
gfg@gfg-Lenovo-G50-80:~$
```

finalize() Method

- Garbage Collector always calls finalize() method just before deletion/destroying the object to perform clean-up activity
- Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection, or we can say resource deallocation
- Syntax,

protected void finalize throws Throwable { }



PUBLIC
When a member of a class is modified by public, then that member can be accessed by any other code

PRIVATE
When a member of a class is specified as private, then that member can only be accessed by other members of its class

PROTECTED
When a member of a class is specified as protected, then that member can only be accessed by inherited class

String class

- Unlike other languages that implement string as character arrays, Java implements strings as objects of type *String*
- Java has methods to compare two strings, search for a substring, concatenate two strings, change the case of string
- String objects can be constructed a number of ways, making it easy to obtain a string when needed
- To create an empty String, call the default constructor
- For example - *String s = new String();*

String functions

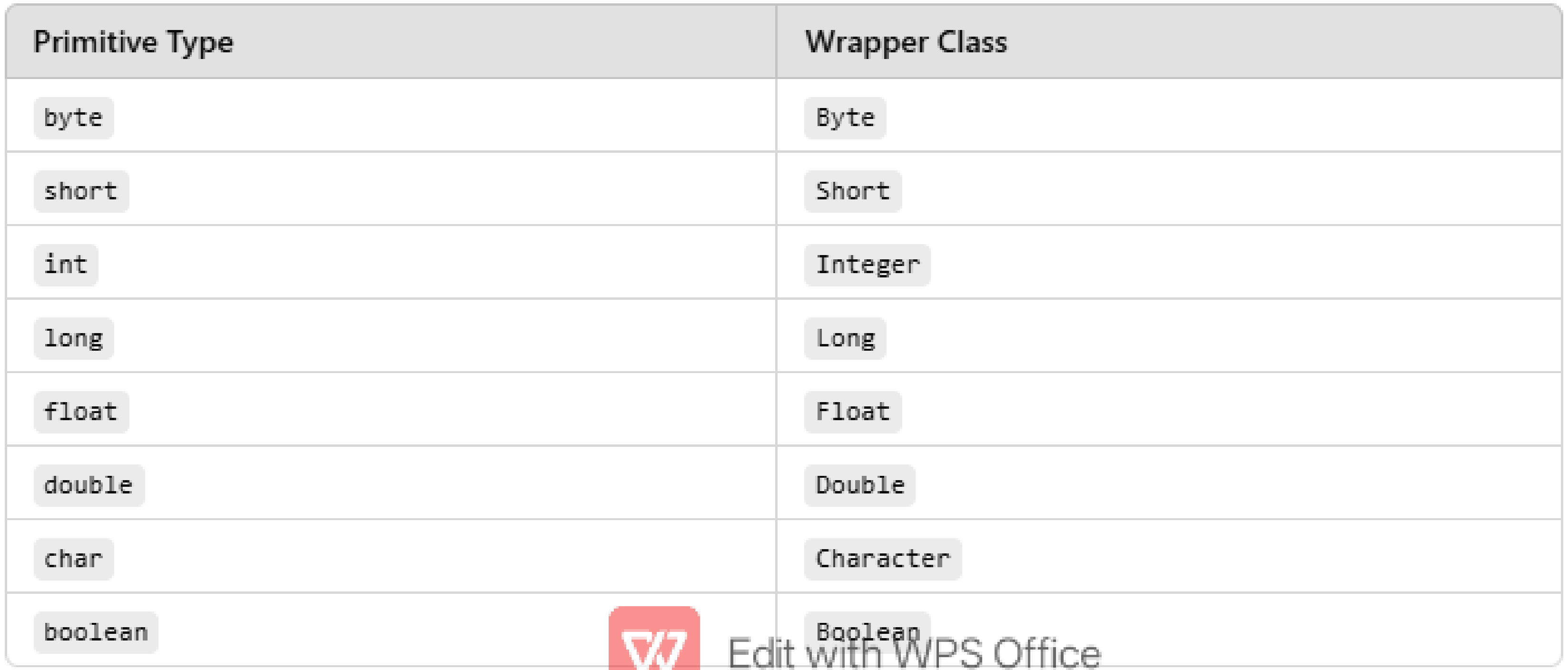
String Method	Description
int length()	Returns the number of characters in the String.
Char charAt(int i)	Returns the character at ith index.
String substring (int i)	Return the substring from the ith index character to end.
String substring (int i, int j)	Returns the substring from i to j-1 index.
String concat(String str)	Concatenates specified string to the end of this string.
boolean equals(Object otherObj)	Compares this string to the specified object.

String functions

String Method	Description
String toLowerCase()	Converts all the characters in the String to lowercase.
String toUpperCase()	Converts all the characters in the String to uppercase.
String trim()	Returns the copy of the String, by removing white spaces at both ends. It does not affect whitespaces in the middle.
String replace (char oldChar, char newChar)	Returns new string by replacing all occurrences of oldChar with newChar.
boolean contains(CharSequence sequence)	Returns true if string contains contains the given string.

Wrapper classes

- **Wrapper classes** convert **primitive data types** into **objects**.
- They are in the java.lang **package**.
- Java provides a wrapper class for each **primitive type**.



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

Thank you