Bootcamp Training

Java Spring AWS Frontend Training

SummitWorks™
GLOBAL SOLUTION ARCHITECTS

Authorized & published by Summitworks Technologies Inc

Agenda: Day -1

- Introduction to Programming
- Platform Independence in Java
  - Compiler and JVM
  - Interpreter and JIT Compiler
  - JVM Architecture
- Structure of Java Program
- Primitive Data Types
- Java Decision making
  - if, If-else and if...else if...else Statement
  - Nested if Statement and Switch Statement
  - ? : Operator
- Java – Loops and Transfer Statements
  - while and do while Loop
  - for and for each loop
  - Loop Control Statements
    - Break Statement
    - Continue Statement
- Enhanced for loop in Java
- Operators
  - The Arithmetic and Relational Operators
  - The Bitwise and Logical Operators
  - The Assignment Operators
  - Precedence of Operators
- Casts and Conversions
  - Widening and Narrowing
- The String Class
  - Creating Strings
  - Immutability of String Object
  - String, String Buffer & String Builder Classes
  - String Length and String Methods
- Method Signatures and Lambda Expressions Java 8
- Arrays
  - Creating and Processing Arrays
  - Copying Arrays and Arrays of Objects
- The import Statement

Training Objective

To Train fresh graduates or experienced professionals in IT Career Paths to facilitate their development towards becoming professionals in the IT Industry

Training Highlights

- Industry standards based with real time tools and concepts
- Harden skills through Hands-On and Project Work
- Flavored with the latest technology in software

Introduction

- Java was developed and created in the early 1990s by Sun Microsystems and founded by James Gosling.

- Java Website:
  https://docs.oracle.com/javase/tutorial/

Introduction

- Java is among the most popular programming languages out there, mainly because of how versatile and compatible it is.

- Java can be used for a large number of things, including software development, mobile applications, and large systems development.

- As of 2019, 88% market share of all smartphones run on Android, the mobile operating system written in Java. Knowing Java opens a great deal of doors for you as a developer.

The Java Programming Language Platforms

There are four platforms of the Java programming language:

- Java Platform, Standard Edition (Java SE)
- Java Platform, Enterprise Edition (Java EE)
- Java Platform, Micro Edition (Java ME)
- JavaFX

All Java platforms consist of a Java Virtual Machine (VM) and an application programming interface (API). The Java Virtual Machine is a program, for a particular hardware and software platform, that runs Java technology applications. An API is a collection of software components that you can use to create other software components or applications. Each Java platform provides a virtual machine and an API, and this allows applications written for that platform to run on any compatible system with all the advantages of the Java programming language

The Java Programming Language Platforms

- **Java Platform, Standard Edition (Java SE):** This is the core Java programming platform. It contains all of the libraries and APIs that any Java programmer should learn (java.lang, java.io, java.math, java.net, java.util, etc...).
- **Java Platform, Enterprise Edition (Java EE):** it adds libraries which provide functionality to deploy fault-tolerant, distributed, multi-tier Java software, based largely on modular components running on an application server.
- **Java Platform, Micro Edition (Java ME):** This is the platform for developing applications for mobile devices and embedded systems such as set-top boxes. Java ME provides a subset of the functionality of Java SE, but also introduces libraries specific to mobile devices.
- **JavaFX:** is a platform for creating rich internet applications using a lightweight user-interface API. It is a recent addition to the family of Java platforms.

Real World Java Applications

There are many places where Java is used in real world, starting from commercial e-commerce website to android apps, from scientific application to financial applications like electronic trading systems, from games like Minecraft to desktop applications like Eclipse, Netbeans, and IntelliJ

Real World Java Applications

Server Apps at Financial Services Industry: Java is very big in Financial Services. Lots of global Investment banks like Goldman Sachs, Citigroup, Barclays, Standard Charted and other banks use Java for writing front and back office electronic trading system, writing settlement and confirmation systems, data processing projects and several others. Java is mostly used to write server side application, mostly without any front end, which receives data form one server process it and sends it other process.

Java Web applications: Java is also big on E commerce and web application space. You have a lot of RESTfull services being created using Spring,. Many of government, healthcare, insurance, education, defense and several other department have their web application built in Java.

Real World Java Applications

Software Tools: Many useful software and development tools are written and developed in Java e.g. Eclipse, InetelliJ Idea and Netbaens IDE.

Trading Application: Third party trading application, which is also part of bigger financial services industry, also use Java. Popular trading application like Murex, which is used in many banks for front to bank connectivity, is also written in Java.

Embedded Space: Java is also big in the embedded space. It shows how capable the platform is, you only need 130 KB to be able to use Java technology (on a smart card or sensor).

Big Data technologies: Hadoop and other big data technologies are also using Java.

Real World Java Applications

Scientific Applications: Nowadays Java is often a default choice for scientific applications, including natural language processing. Main reason of this is because Java is more safe, portable, maintainable and comes with better high-level concurrency tools than C++ or any other language.

Android Apps: If you want to see where Java is used , open your Android phone and any app, they are actually written in Java programming language, with Google's Android API, which is similar to JDK. By the way android uses different JVM and different packaging, but code is still written in Java.

Java Environment

- Java includes many development tools, classes and methods
  - Development tools are part of Java Development Kit (JDK) and
  - The classes and methods are part of **Java Standard Library** (JSL), also known as **Application Programming Interface** (**API**).
- JDK constitutes of tools like java compiler, java interpreter and many.
- API includes hundreds of classes and methods grouped into several packages according to their functionality.
- The first version of java is 1.0 and latest version is 12.0

Java SE API

- Java SE 8 API Documentation

https://docs.oracle.com/javase/8/docs/api/

JDK, JRE and JVM

- JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

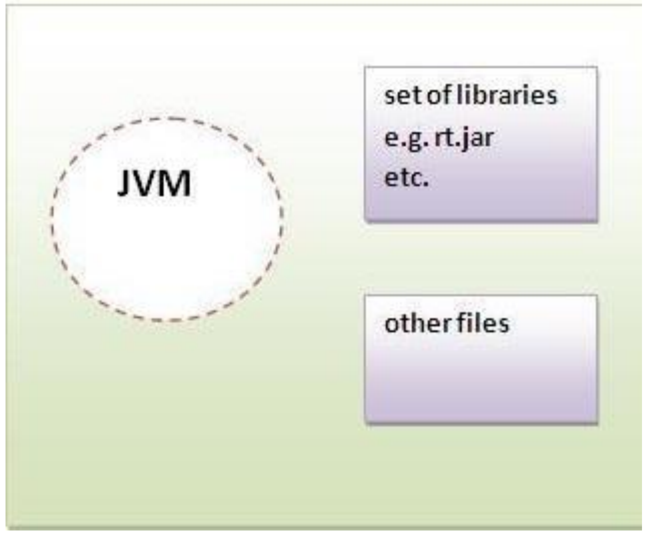
JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

The JVM performs following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

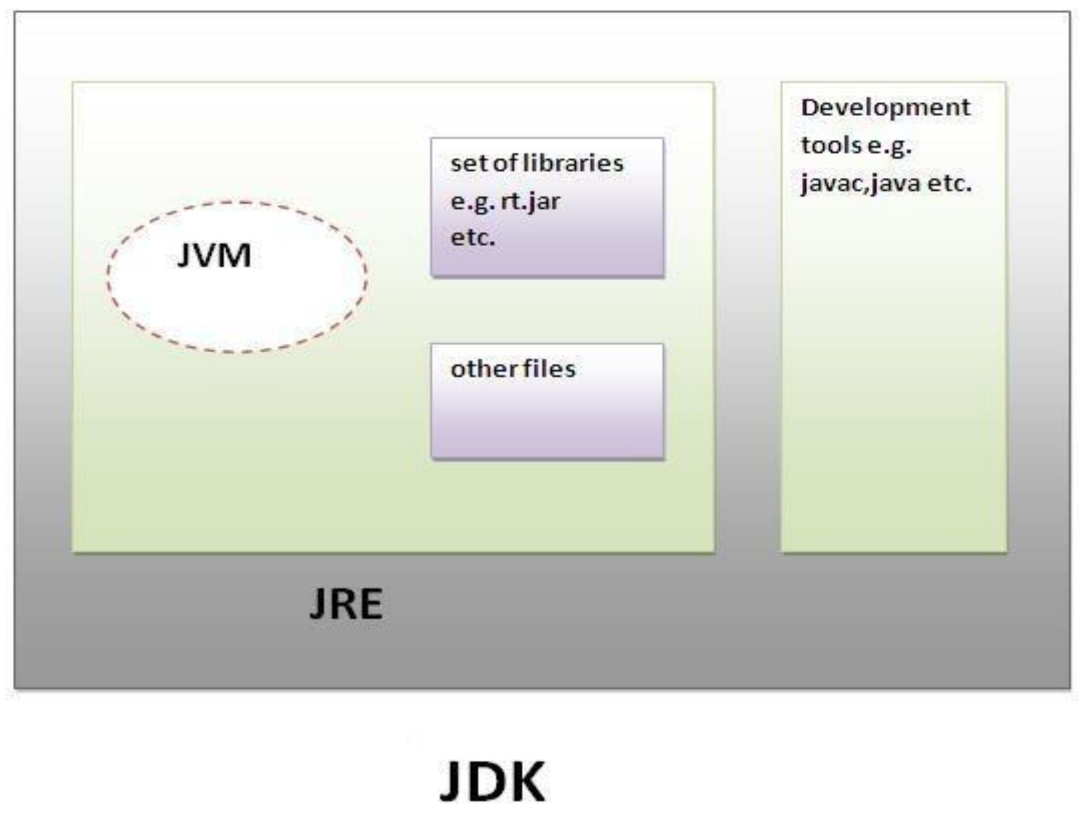
JRE[Java Runtime Environment]

- It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.
- Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.

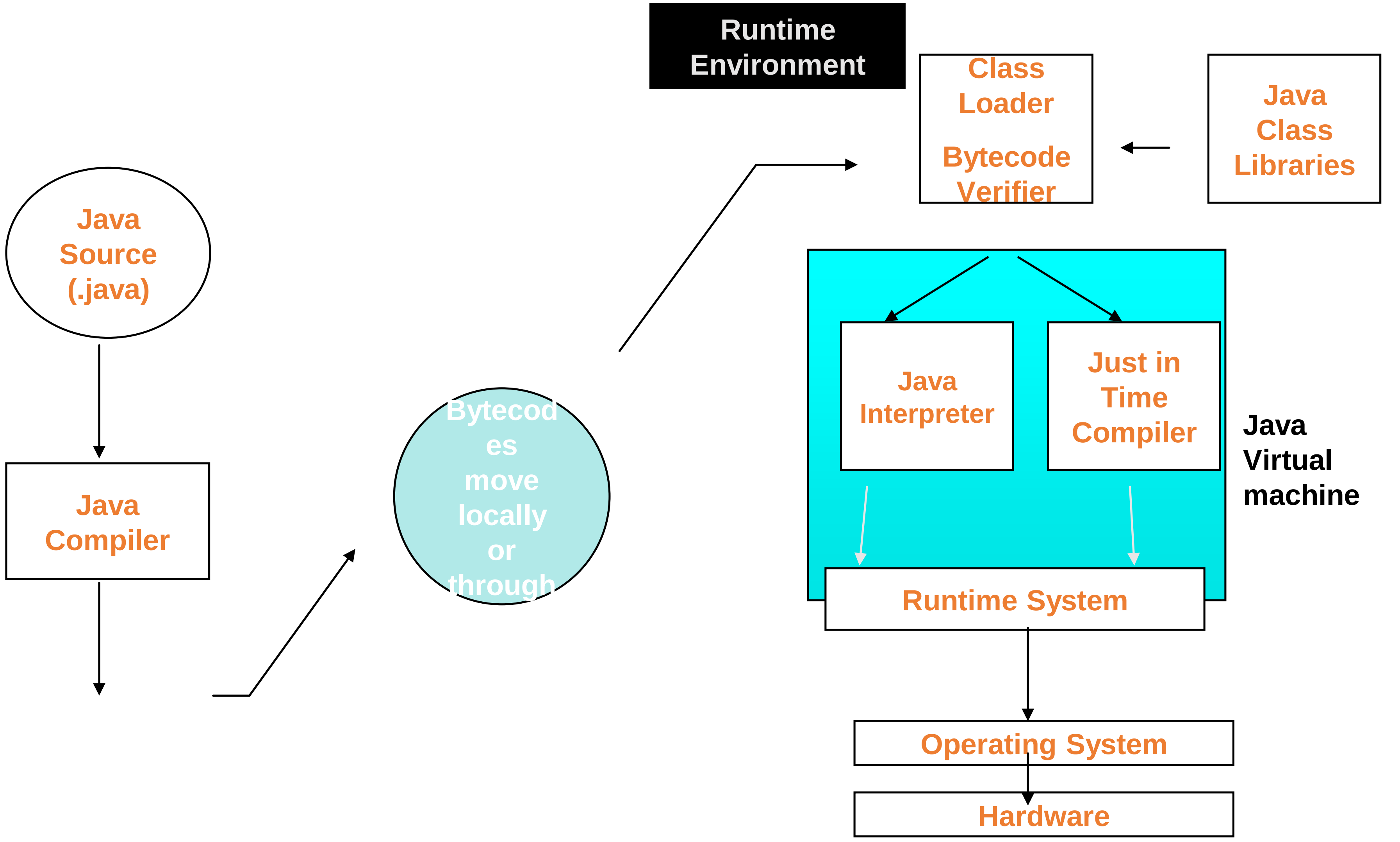


JDK [Java Development Kit]

- It physically exists. It contains JRE + development tools like java compiler.


JVM
set of libraries e.g. rt.jar etc.
other files
Development tools e.g. javac,java etc.
JRE
JDK

Execution Life Cycle of Java Code



Just in Time Compiler (JIT)

- JIT compiles the code when it is needed but not before runtime. Whenever a program is executed this compiled object code is invoked instead of interpreting the entire byte code and is quite efficient. This increases the performance of the program as well. Just in time compiler coverts the byte code to a platform specific executable code that can be executed immediately.

Overview of How JIT Compiler

- Using a compiler, Java source code is converted to Java byte code (.class files).
- Once this is done, JVM loads the .class files at run time and converts them to a machine understandable code using an interpreter.
- JIT compiler is a feature of JVM which when enabled makes the JVM analyze the method calls in byte code and compiles them to more native and efficient code. JIT optimizes the prioritized method calls at this point of time.
- Once these method calls are compiled, the JVM then executes this optimized code instead of interpreting it which is likely to increase the performance of the execution.

Characteristics of Java

- ✓ Java is simple
- ✓ Java is High Level
- ✓ Java is Powerful
- ✓ Java is object-oriented
- ✓ Java is distributed
- ✓ Java is interpreted
- ✓ Java is robust
- ✓ Java is Compiler & Interpreter
- ✓ Java is portable & platform
- ✓ Java is high performance
- ✓ Java is multithreaded
- ✓ Java is dynamic
- ✓ Java is secure & safe

Java Virtual Machine

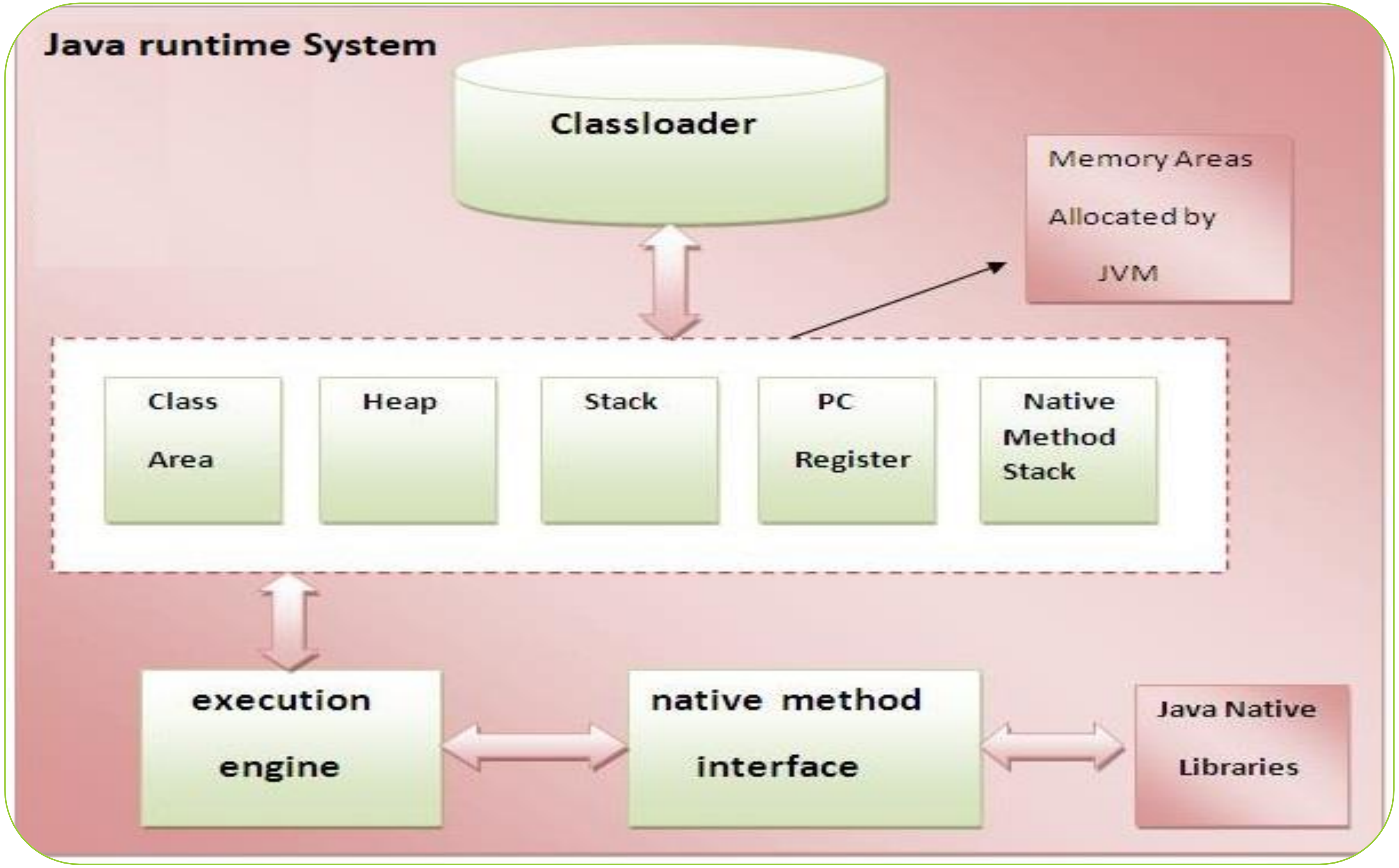
The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting
- Security , etc.

Architecture of JVM



Elements of JVM Architecture

1) Classloader : Class loader is a subsystem of JVM that is used to load class files.

2) Class(Method) Area : Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap: It is the runtime data area in which objects are allocated.

4) Stack: Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register: PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

Elements of JVM Architecture

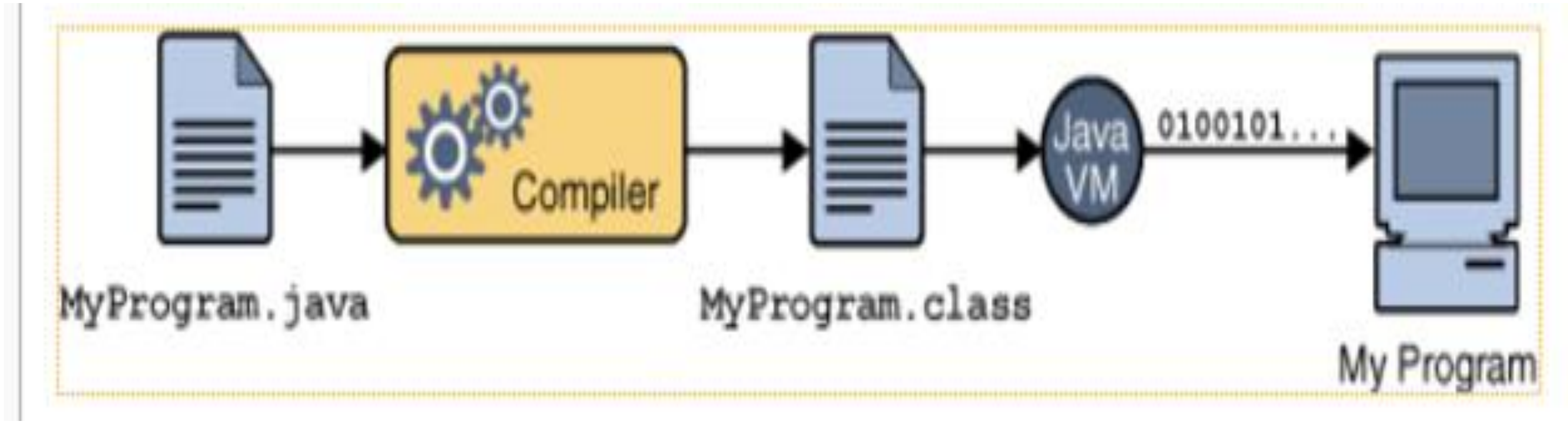
6) Native Method Stack: It contains all the native methods used in the application.

7) Execution Engine

It contains:

- A virtual processor
- Interpreter: Read bytecode stream then execute the instructions.
- Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

WORA(Write Once Run Anywhere)`



Importance of Bytecode ?

- ✓Platform-independent
- ✓Loads from the Internet faster than source code
- ✓Interpreter is faster and smaller than it would be for Java source
- ✓Source code is not revealed to end users
- ✓Interpreter performs additional security checks, screens out malicious code

Java Development Tools

- Java Editors ., ex: Edit plus, Notepad++., etc.,

- IDE (Integrated Development Environment)
    1. Eclipse
    2. My Eclipse
    3. Net Beans
    4. RAD
    5. etc.,

- Usually JDK is installed separately and an IDE is installed on top of it.

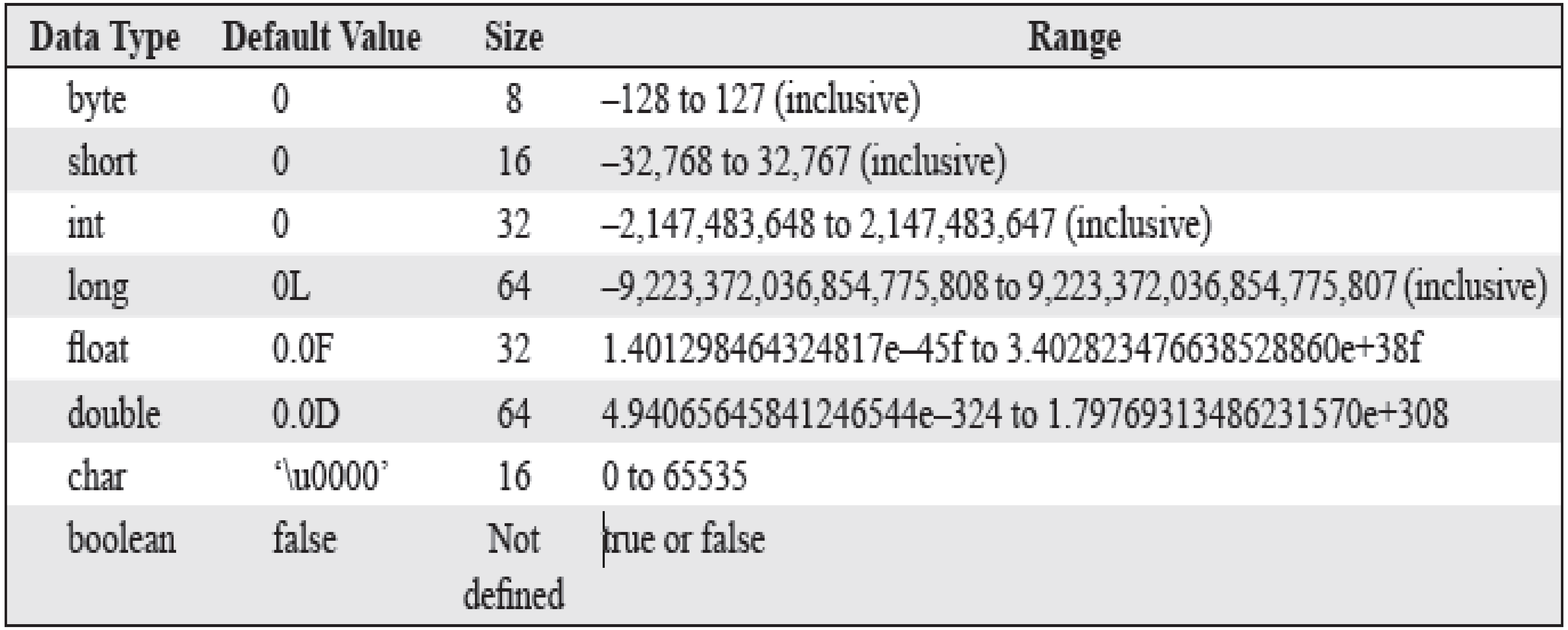
Variables

- a symbolic name
- used to store value which can change during execution of the program.
- int noOfWatts = 100; // variable declaration
- declaration involves specifying the
  - type (Data Type),
  - name (Identifier) and
  - value (Literal) according to the type of the variable.

Data types

- Two types
  - Primitive data types: built in types
  - Non primitive data types: user defined types

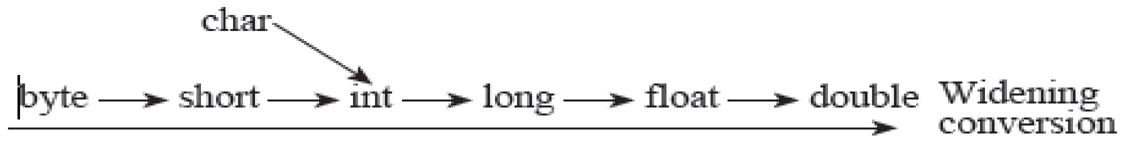
Primitive Data types default values and their range



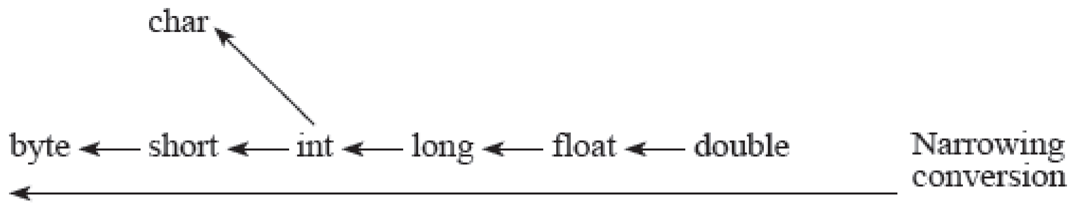
Data Type	Default Value	Size	Range
byte	0	8	–128 to 127 (inclusive)
short	0	16	–32,768 to 32,767 (inclusive)
int	0	32	–2,147,483,648 to 2,147,483,647 (inclusive)
long	0L	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (inclusive)
float	0.0F	32	1.401298464324817e–45f to 3.402823476638528860e+38f
double	0.0D	64	4.940656458412465442e–324 to 1.79769313486231570e+308
char	'\u0000'	16	0 to 65535
boolean	false	Not defined	true or false

Conversion and Casting

- Conversions are performed automatically
  - For e.g. a smaller box can be placed in a bigger box and so on.



- **Casting** also known as narrowing conversion (reverse of widening conversion).
  - A bigger box has to be placed in a small box.
  - Casting is not implicit in nature.
  - Use casting operator i.e. ()
  - int i = (int)(8.0/3.0);



Comments

Java supports three types of comments –

1. /* text */
- The compiler ignores everything from /* to */

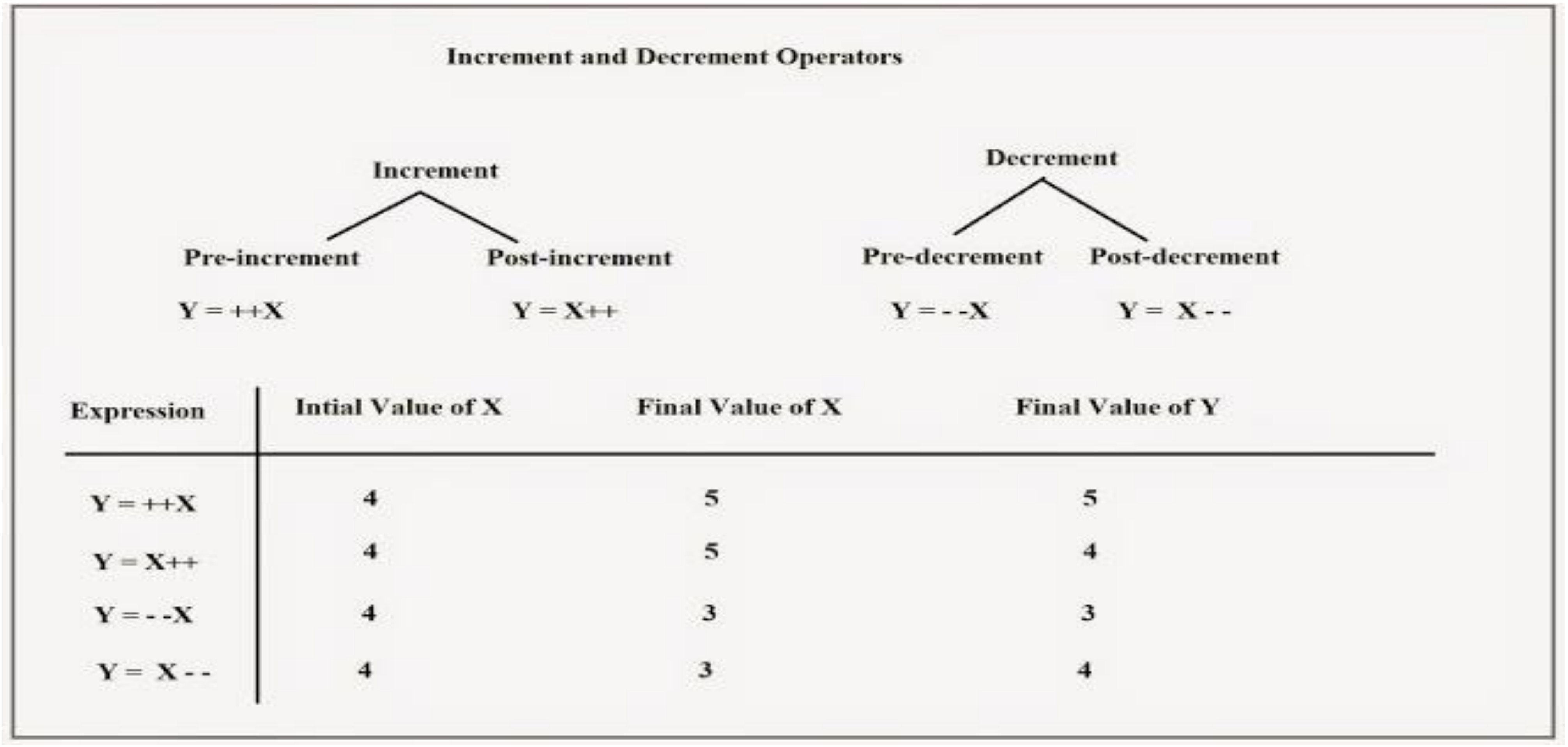
2 //text
- The compiler ignores everything from // to the end of the line.

3 /** documentation */
- This is a documentation comment and in general its called doc comment. The JDK javadoc tool uses doc comments when preparing automatically generated documentation.

- **Javadoc Tool:** Javadoc is a tool for generating API documentation in HTML format from doc comments in source code.

  http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html

Increment and Decrement operators


Increment and Decrement Operators

Expression	Intial Value of X	Final Value of X	Final Value of Y
Y = ++X	4	5	5
Y = X++	4	5	4
Y = - -X	4	3	3
Y = X - -	4	3	4

Logical Operators

Operator	Name	Type	Description
!	Not	Unary	Returns true if the operand to the right evaluates to false. Returns false if the operand to the right is true.
&	And	Binary	Returns true if both of the operands evaluate to true. Both operands are evaluated before the And operator is applied.
|	Or	Binary	Returns true if at least one of the operands evaluates to true. Both operands are evaluated before the Or operator is applied.
^	Xor	Binary	Returns true if one — and only one — of the operands evaluates to true. Returns false if both operands evaluate to true or if both operands evaluate to false.
&& **[Short Circuit Logical Operators]**	Conditional And	Binary	Same as &, but if the operand on the left returns false, it returns false without evaluating the operand on the right.
|| **[Short Circuit Logical Operators]**	Conditional Or	Binary	Same as |, but if the operand on the left returns true, it returns true without evaluating the operand on the right.

Relational Operators

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

Named constants

- A variable is a named memory location that can hold a value of a specific data type; as we have seen, the value stored at this location can change throughout the execution of a program
- If we want to maintain a value in a named location, we use the Java keyword ***final*** in the declaration and immediately assign the desired value; with this mechanism, we declare a named constant. Some examples:

```
final int LUCKY = 7;
final double PI = 3.14159;
final double LIGHTSPEED = 3.0e10.0;
```

Control Statements

- Selection Statements
  - If and switch
- Iteration Statements
  - While, do-while, for and nested loops
- Jump Statements
  - Break, continue and return

Selection Statements - If

- If (condition) statement1; else statement2
  If condition is true, it will execute statement1 else statement2.
- Example

```
int a = 3, b = 4;
if (a >b) a =1;  // since a>b is false, it
  else b = 1      // will set b = 1
```

Selection Statement - switch

- It is a multiway branch statement. It is similar to if-else-is, but is more well organized.

```
Switch (expression) {
    Case value1:  //statement1
    Break;
    Case value2: //satement2
    Break
    …
    Default;
    //default statement
}
```

if expression is value1, it will jump to statement 1 until it hits break

Iteration Statements

- While
- Do-while
- For
- Nested loop

Iteration Statements - While

```
while(condition)
   {
   // statements to keep executing while condition is true
   ..
   ..
   }
```

Example

```
   //Increment n by 1 until n is greater than 100
   while (n > 100) {
   n = n + 1;
   }
```

Iteration Statements – do while

```
Do {
    // statements to keep executing while condition is true
} while(condition)
```

It will first executes the statement and then evaluates the condition.

Example

```
int n = 5;
Do {
System.out.println(" n = " + n);
N--;
    } while(n > 0);
```

Iteration Statements – for

```
 for(initializer; condition; incrementer)
{
// statements to keep executing while condition is true
}
```

Example

```
int i;
int length = 10;
for (i = 0; i < length; i++) {
. . .
// do something to the (up to 9 )
. . .
}
```

Multiple Initializers and Increments

- Sometimes it's necessary to initialize several variables before beginning a for loop.
- Similarly you may want to increment more than one variable.
- Example

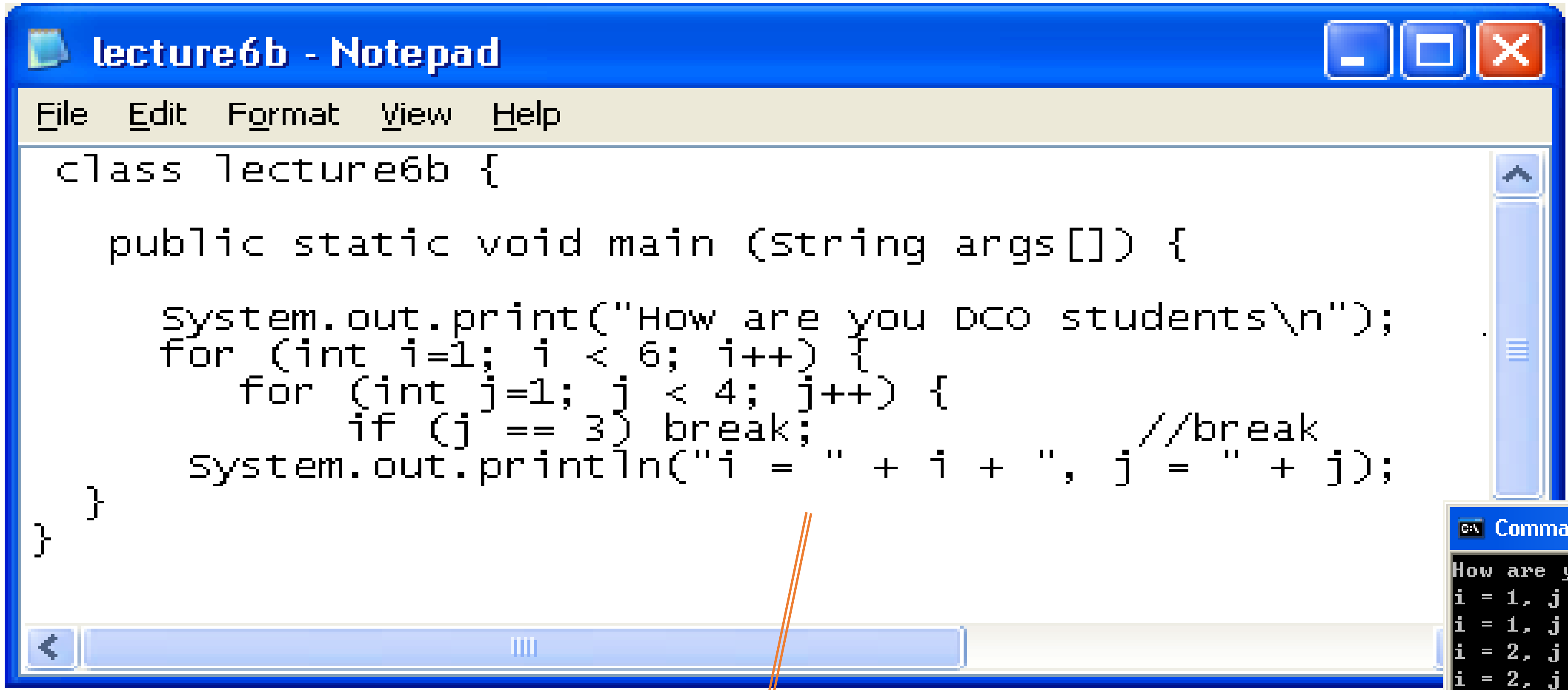
```
for (int i = 1, j = 100; i < 100; i = i+1, j = j-1)
{ System.out.println(i + j); }
```

Jump Statement

- Three jump statements

  Break, continue and return

Break and Continue with the nested loop

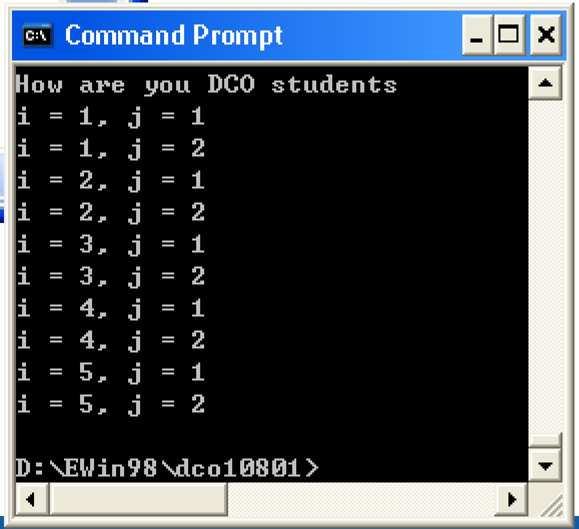


```
class lecture6b {

   public static void main (String args[]) {

      System.out.print("How are you DCO students\n");
      for (int i=1; i < 6; i++) {
         for (int j=1; j < 4; j++) {
             if (j == 3) break;              //break
       System.out.println("i = " + i + ", j = " + j);
    }
}
```

result: j 1 and 2) only



```
How are you DCO students
i = 1, j = 1
i = 1, j = 2
i = 2, j = 1
i = 2, j = 2
i = 3, j = 1
i = 3, j = 2
i = 4, j = 1
i = 4, j = 2
i = 5, j = 1
i = 5, j = 2

D:\EWin98\dco10801>
```

break

Continue - Explanation

- Sometimes it is necessary to exit from the middle of a loop.
- Sometimes we want to start over at the top of the loop.
- Sometimes we want to leave the loop completely. For Example

```
for (int i = 0; i < length; i++) {
if (m[i] % 2 == 0) continue;
// process odd elements... }
```

it will process
the odd case

Return

- It has purpose to exit from the current method
- It jumps back to the statement within the calling method that follows the original method call.
- Example

return expression

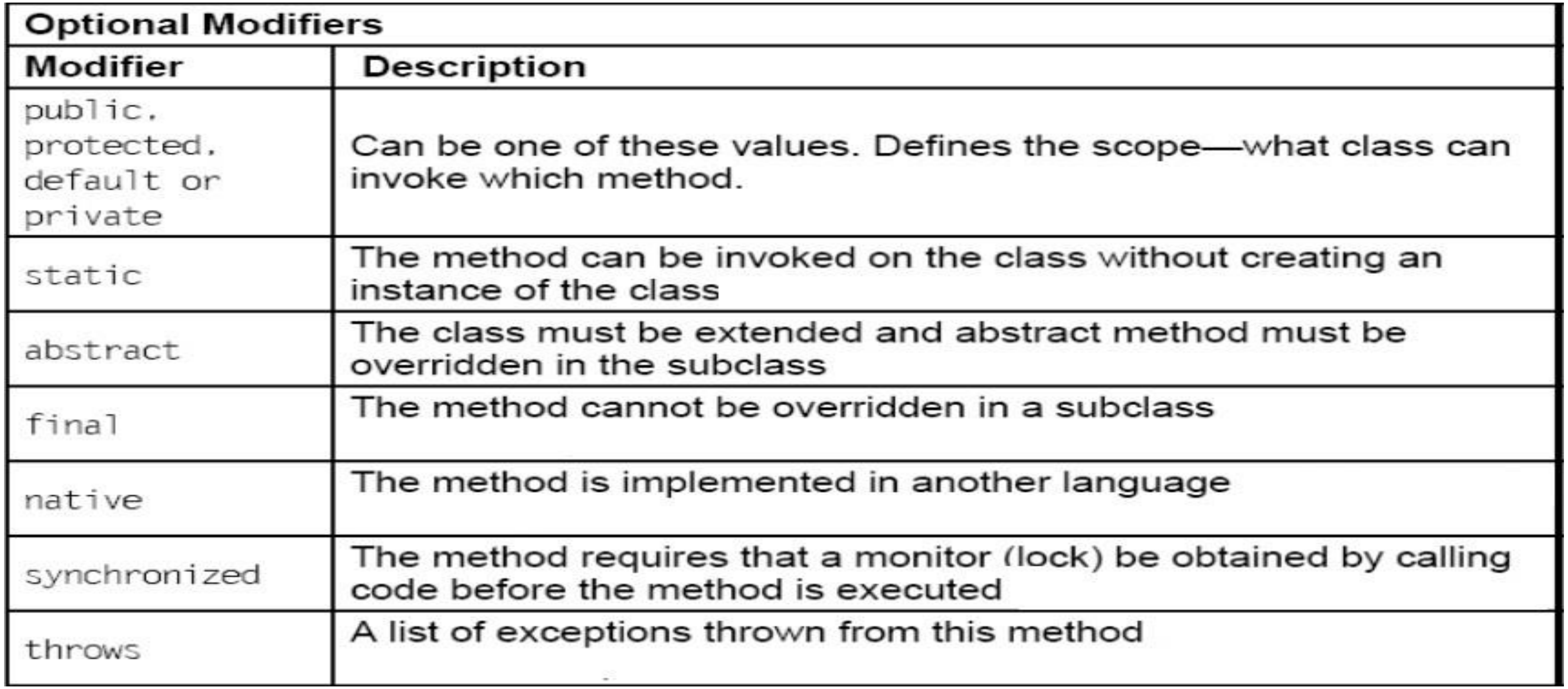
Methods

- similar to a function in any other programming languages.
- None of the methods can be declared outside the class.
- Why use methods?
  - To make code reusable
  - To parameterize code
  - For top-down programming
  - To simplify code

Method (contd.)

- General syntax for a method declaration:
  [modifiers] return_type method_name (parameter_list) [throws_clause] { [statement_list]}
- The method declaration includes:
  - Modifiers: The modifiers are optional. They can be, public, protected, default or private, static, abstract, final, native, synchronized, throws

Optional Modifiers Used with Methods



Optional Modifiers	
Modifier	**Description**
`public`, `protected`, `default` or `private`	Can be one of these values. Defines the scope—what class can invoke which method.
`static`	The method can be invoked on the class without creating an instance of the class
`abstract`	The class must be extended and abstract method must be overridden in the subclass
`final`	The method cannot be overridden in a subclass
`native`	The method is implemented in another language
`synchronized`	The method requires that a monitor (lock) be obtained by calling code before the method is executed
`throws`	A list of exceptions thrown from this method

Method (contd.)

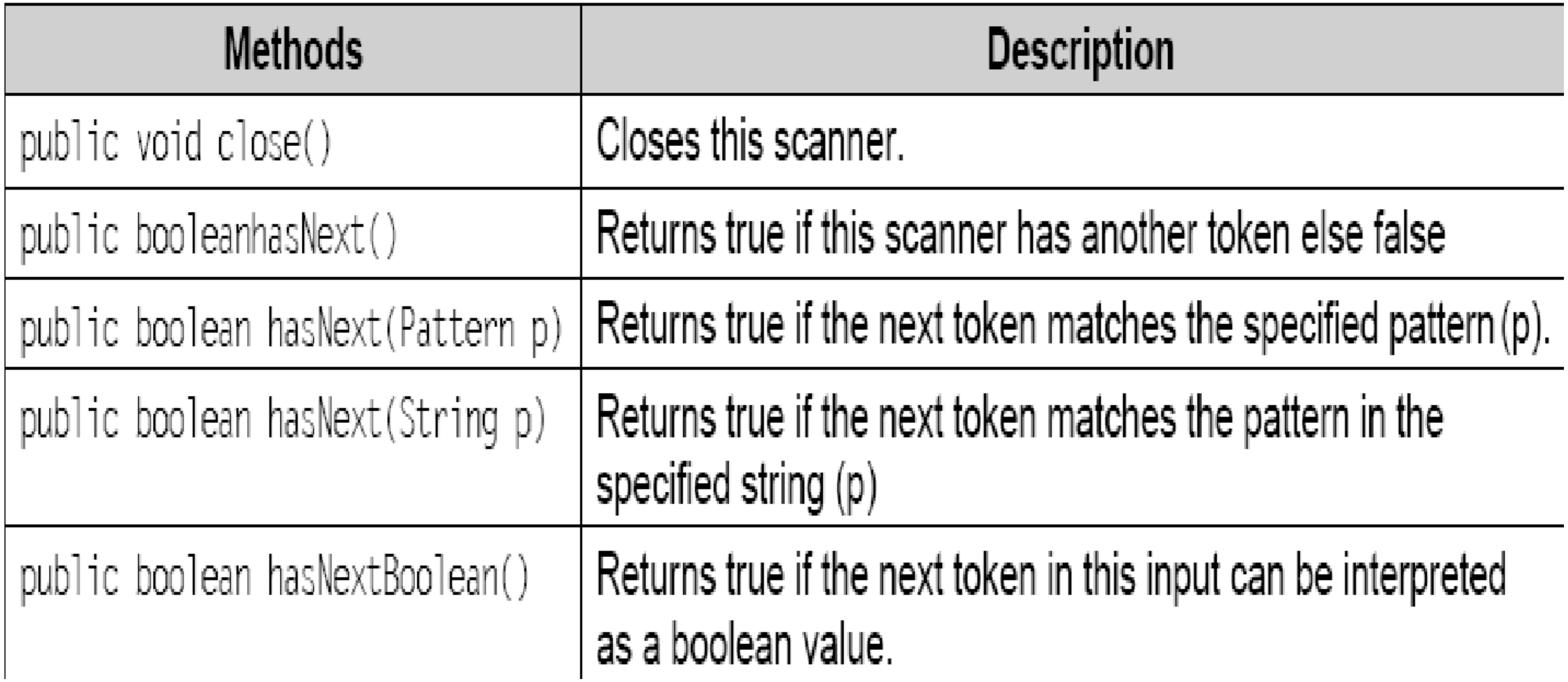
- Return Type:
  - can be either void or if a value is returned, it can be either a primitive type or a class.
  - If the method declares a return type, then before it exits it must have a return statement.
- Method Name**:**
  - The method name must be a valid Java identifier.

  Parameter List**:**
  - contains zero or more type/identifier pairs make up the parameter list.
  - Each parameter in parameter list is separated by a comma.

java.util.Scanner class

- Prior to JDK 6, JDK 5 introduced the *Scanner* class (java.util package) which can be used for getting input from user (both lines of text as well as primitives)
- Can also be used for breaking the input String into tokens separated by a delimiter which is by default a white space.

Methods of Scanner class



Methods	Description
public void close()	Closes this scanner.
public booleanhasNext()	Returns true if this scanner has another token else false
public boolean hasNext(Pattern p)	Returns true if the next token matches the specified pattern(p).
public boolean hasNext(String p)	Returns true if the next token matches the pattern in the specified string (p)
public boolean hasNextBoolean()	Returns true if the next token in this input can be interpreted as a boolean value.

String class facts

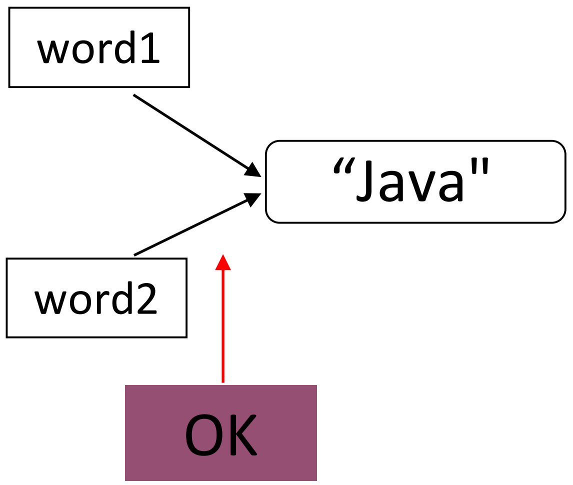
- An object of the String class represents a string of characters.
- The String class belongs to the java.lang package, which does not require an import statement.
- Like other classes, String has constructors and methods.
- Unlike other classes, String has two operators, + and += (used for concatenation).

Ways of creating string class object

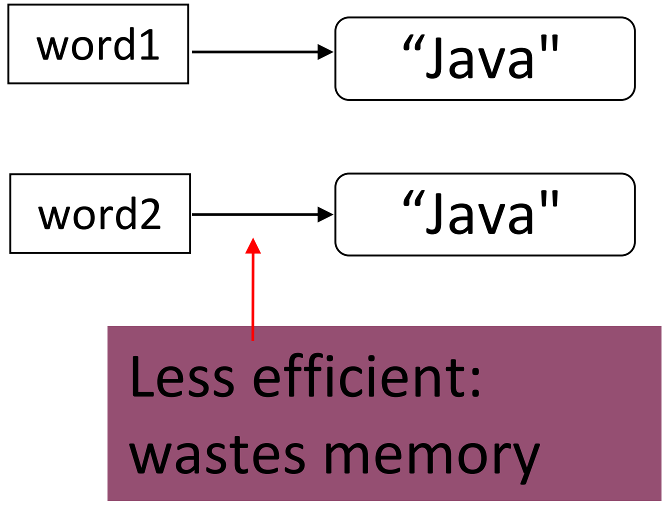
- What is the difference between
  - String s=“hello”;
  - String s=new String(“hello”);

Uses less memory.

```
String word1 = "Java";
String word2 = word1;
```



```
String word1 = “Java";
String word2 = new String(word1);
```



String class methods

- char charAt(int index)
  Returns the character at the specified index.

- int compareTo(Object o)
  Compares this String to another Object.

- int compareTo(String anotherString)
  Compares two strings lexicographically.

- int compareToIgnoreCase(String str)
  Compares two strings lexicographically, ignoring case differences.

- String concat(String str)
  - Concatenates the specified string to the end of this string.

Immutability

- What is the result?

```
String s=“Hello”;
s.concat(“ Hai”);
System.out.println(s);
```

Immutability

- Once created, a string cannot be changed: none of its methods changes the string.
- Such objects are called *immutable*.

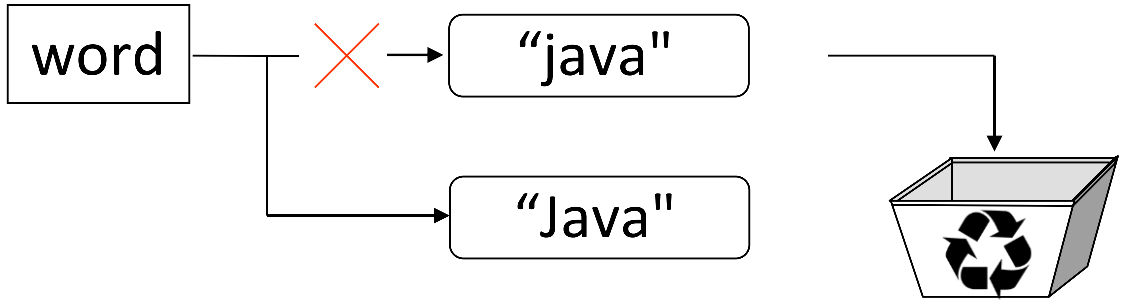
Advantage of Immutability

- Immutable objects are convenient because several references can point to the same object safely: there is no danger of changing an object through one reference without the others being aware of the change

Disadvantages of Immutability

Less efficient — you need to create a new string and throw away the old one even for small changes.

```
String word = “Java";
char ch = Character.toUpperCase(word.charAt (0));
word =  ch + word.substring (1);
```



String

- Strings in java are immutable
- Once created they cannot be altered and hence any alterations will lead to creation of new string object

StringBuffer

- StringBuffer is a synchronized and allows us to mutate the string.
- StringBuffer has many utility methods to manipulate the string.
- This is more useful when using in a multithreaded environment.
- Always has a locking overhead.

StringBuilder

- StringBuilder is the same as the StringBuffer class
- The StringBuilder class is not synchronized and hence in a single threaded environment, the overhead is less than using a StringBuffer.

Methods

- Similar to a function in any other programming languages.
- None of the methods can be declared outside the class.
- Why use methods?
    - To make code reusable
    - To parameterize code
    - For top-down programming
    - To simplify code

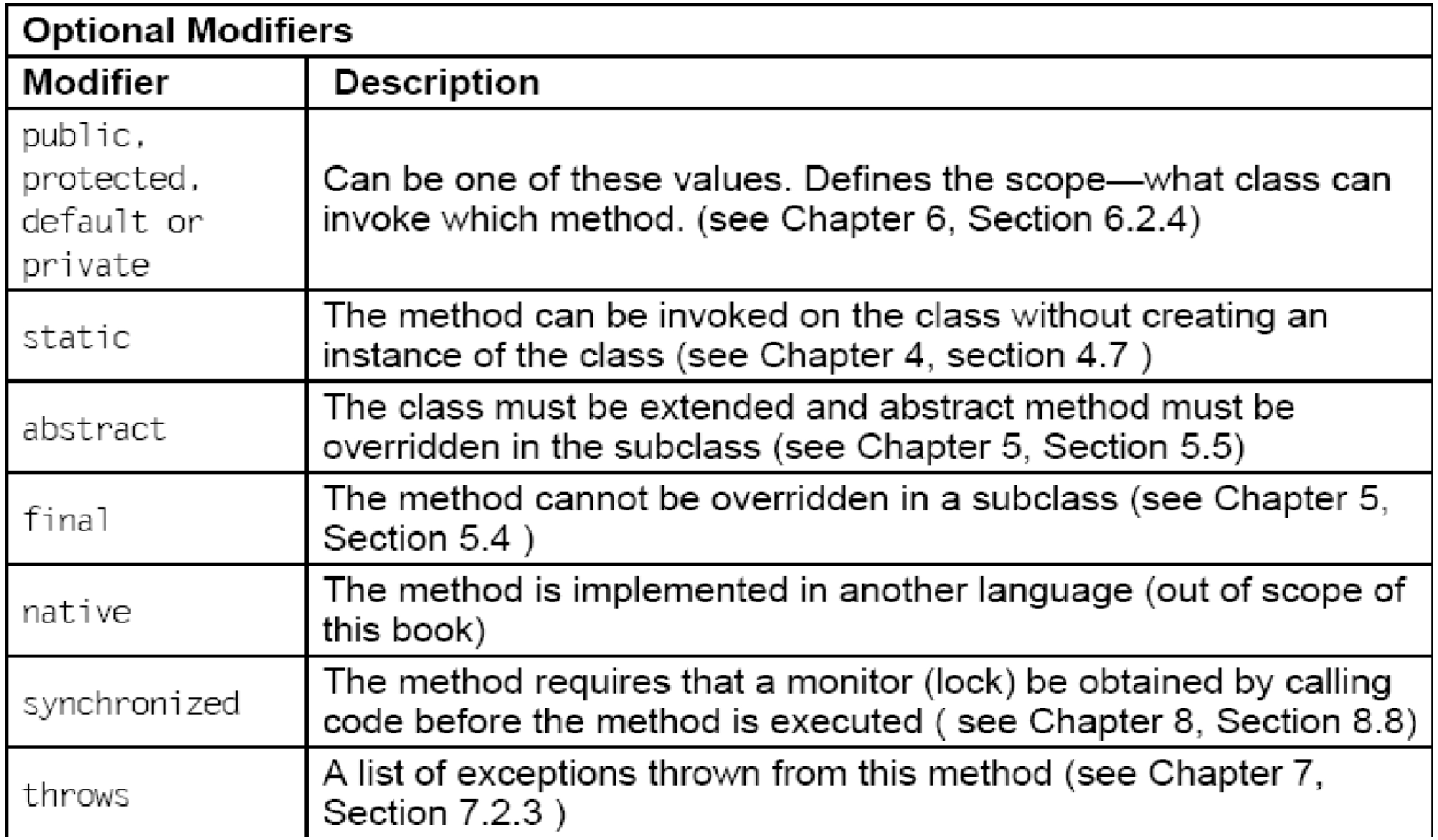
Method (contd.)

- General syntax for a method declaration:

```
[modifiers] return_type method_name (parameter_list) [throws_clause]
{   [statement_list]
}
```

- The method declaration includes:
    - Modifiers: The modifiers are optional. They can be, public, protected, default or private, static, abstract, final, native, synchronized, throws

Optional Modifiers Used with Methods



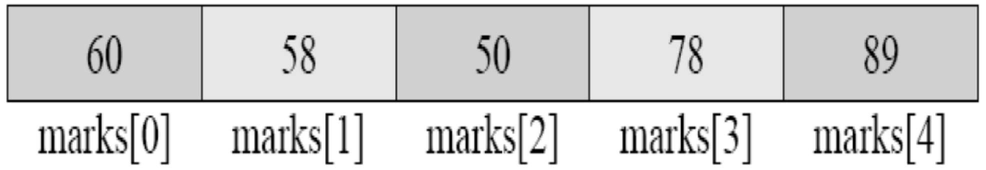
Optional Modifiers

Modifier	Description
`public`, `protected`, `default` or `private`	Can be one of these values. Defines the scope—what class can invoke which method. (see Chapter 6, Section 6.2.4)
`static`	The method can be invoked on the class without creating an instance of the class (see Chapter 4, section 4.7)
`abstract`	The class must be extended and abstract method must be overridden in the subclass (see Chapter 5, Section 5.5)
`final`	The method cannot be overridden in a subclass (see Chapter 5, Section 5.4)
`native`	The method is implemented in another language (out of scope of this book)
`synchronized`	The method requires that a monitor (lock) be obtained by calling code before the method is executed (see Chapter 8, Section 8.8)
`throws`	A list of exceptions thrown from this method (see Chapter 7, Section 7.2.3)

Array

- There are situations where we might wish to store a group of similar type of values in a variable.
- Array is a memory space allocated, which can store multiple values of same data type, in contiguous locations.
- This memory space, which can be perceived to have many logical contiguous locations, can be accessed with a common name.

1-D Array



```
Marks[0] = 60;
Marks[1] = 58;
Marks[2] = 50;
Marks[3] = 78;
Marks[4] = 89;
```

Creation of Arrays

- Creating an array, similar to an object creation, can inherently involve three steps:
  - Declaring an array
  - Creating memory locations
  - Initializing/assigning values to an array

Declaring an array

- Declaring an array is same as declaring a normal variable except that you must use a set of square brackets with the variable type.
- There can betwo ways in which an array can be declared.
  - type arrayname[ ];
  - type[ ] arrayname;
- For e.g.

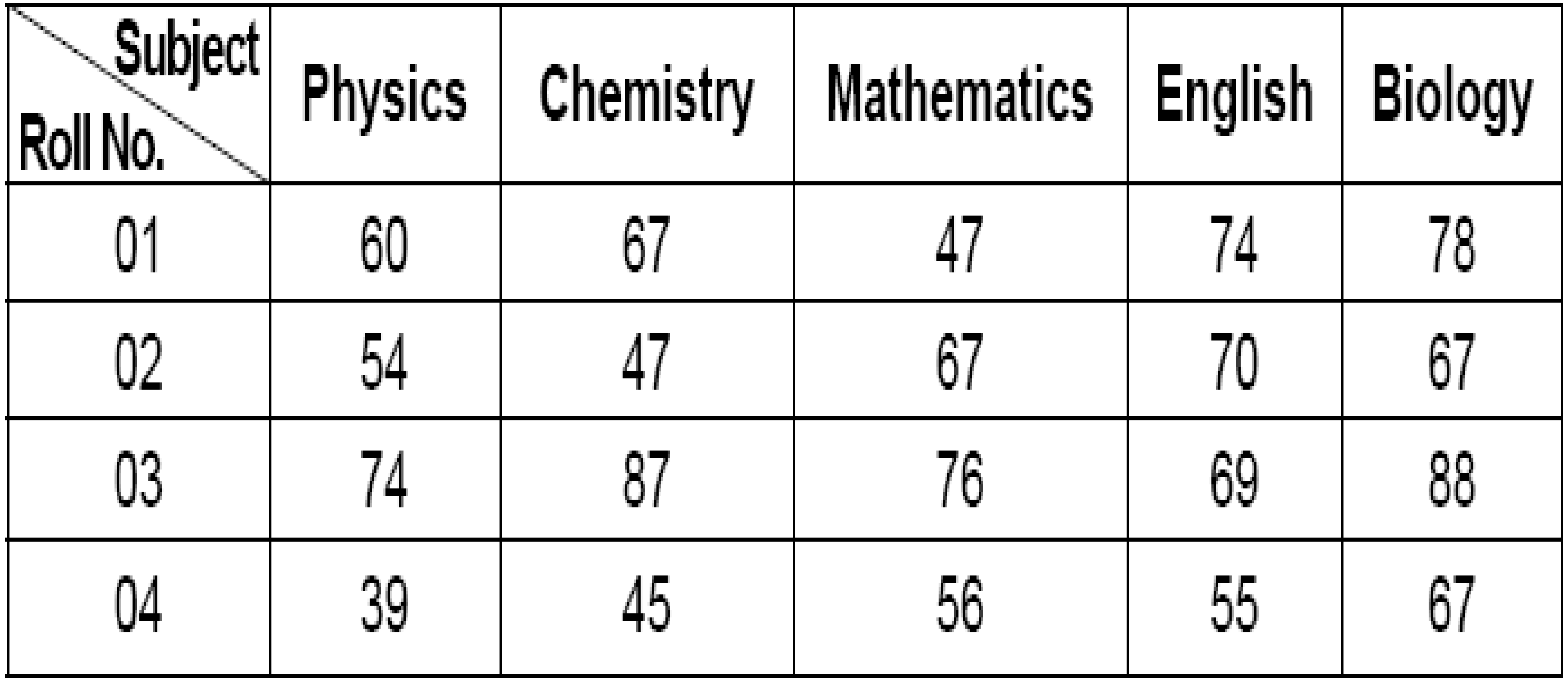
  int marks[ ]; or

  int[ ] marks;

2-D Arrays (contd.)

- A two-dimensional array can have values that go not only across but also across.
- Sometimes values can be conceptualized in the form of table that is in the form of rows and columns.
- Suppose we want to store the marks of different subjects. We can store it in a one-dimensional array.

2-D Arrays (contd.)



Subject / Roll No.	Physics	Chemistry	Mathematics	English	Biology
01	60	67	47	74	78
02	54	47	67	70	67
03	74	87	76	69	88
04	39	45	56	55	67

Using for-each with arrays

The enhanced for loop, i.e. for-each was introduced in Java 5 to enhance iterations over arrays and other collections easier. The format of for-each is as follows:

- for (type var : arr){
- // Body of loop
- }

Any queries