CSE310: Programming in Java

Fundamentals of Programming in Java

Naming Conventions

- Use lowercase for variables and methods. If a name consists of several words, concatenate them into one, making the first word lowercase and capitalizing the first letter of each subsequent word—for example, the variables radius and area and the method print.
- Capitalize the first letter of each word in a class name—for example, the class names : ComputeArea and System.
- Capitalize every letter in a constant, and use underscores between words—for example, the constants PI and MAX_VALUE.

It is important to follow the naming conventions to make your programs easy to read.

Identifiers

- A name in a program is called an identifier.
- Identifiers can be used to denote classes, methods, variables, and labels.
- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- An identifier must start with a letter, an underscore (**_**), or a dollar sign (**$**). It cannot start with a digit.

  Example: number, Number, sum_$, bingo, $$_100

Keywords

Keywords are reserved identifiers that are predefined in the language.

Cannot be used as names for a variable, class, or method.

All the keywords are in lowercase.

There are 50 keywords currently defined in the Java language.

The keywords **const** and **goto** are reserved but not used.

true, false, and **null** are also reserved.

Java Keywords

The following fifty keywords are reserved for use by the Java language:

abstract	double	int	super
assert	else	interface	switch
boolean	enum	long	synchronized
break	extends	native	this
byte	final	new	throw
case	finally	package	throws
catch	float	private	transient
char	for	protected	try
class	goto	public	void
const	if	return	volatile
continue	implements	short	while
default	import	static	
do	instanceof	strictfp*	

The keywords goto and const are C++ keywords reserved, but not currently used in Java. This enables Java compilers to identify them and to produce better error messages if they appear in Java programs.

The literal values true, false, and null are not keywords, just like literal value 100. However, you cannot use them as identifiers, just as you cannot use 100 as an identifier.

In the code listing, we use the keyword color for true, false, and null to be consistent with their coloring in Java IDEs.

Numeric literals

A literal is a constant value that appears directly in a program.

Integer literals(2,98,-45 etc.)

Floating point literals(2.34f,4.675d)

Scientific notations(1.2345e2,1.5678e-2)

Other literals

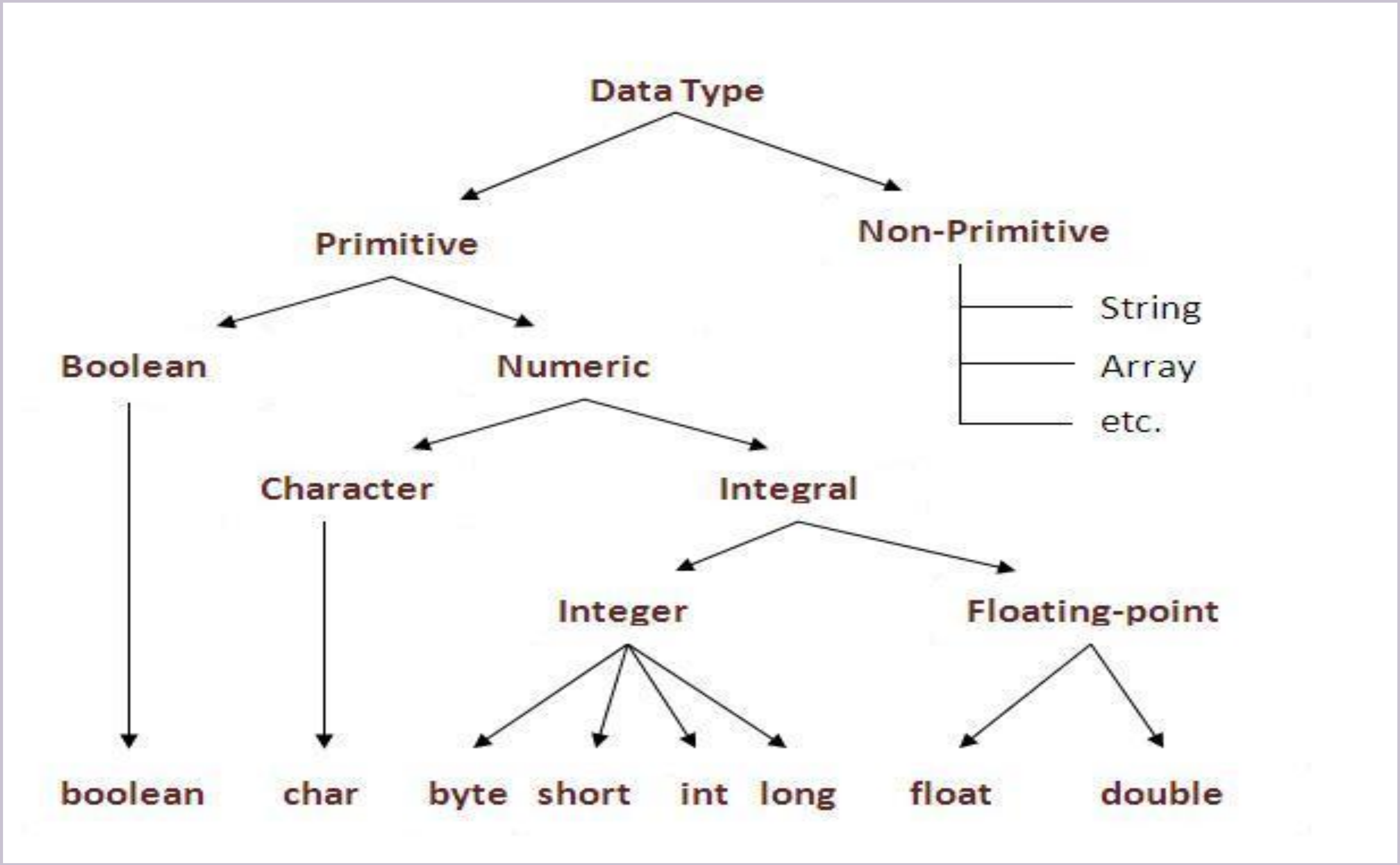
Boolean literals: true and false

Character literals: Always enclosed in single quotes, e.g. ‘A’, ’@’

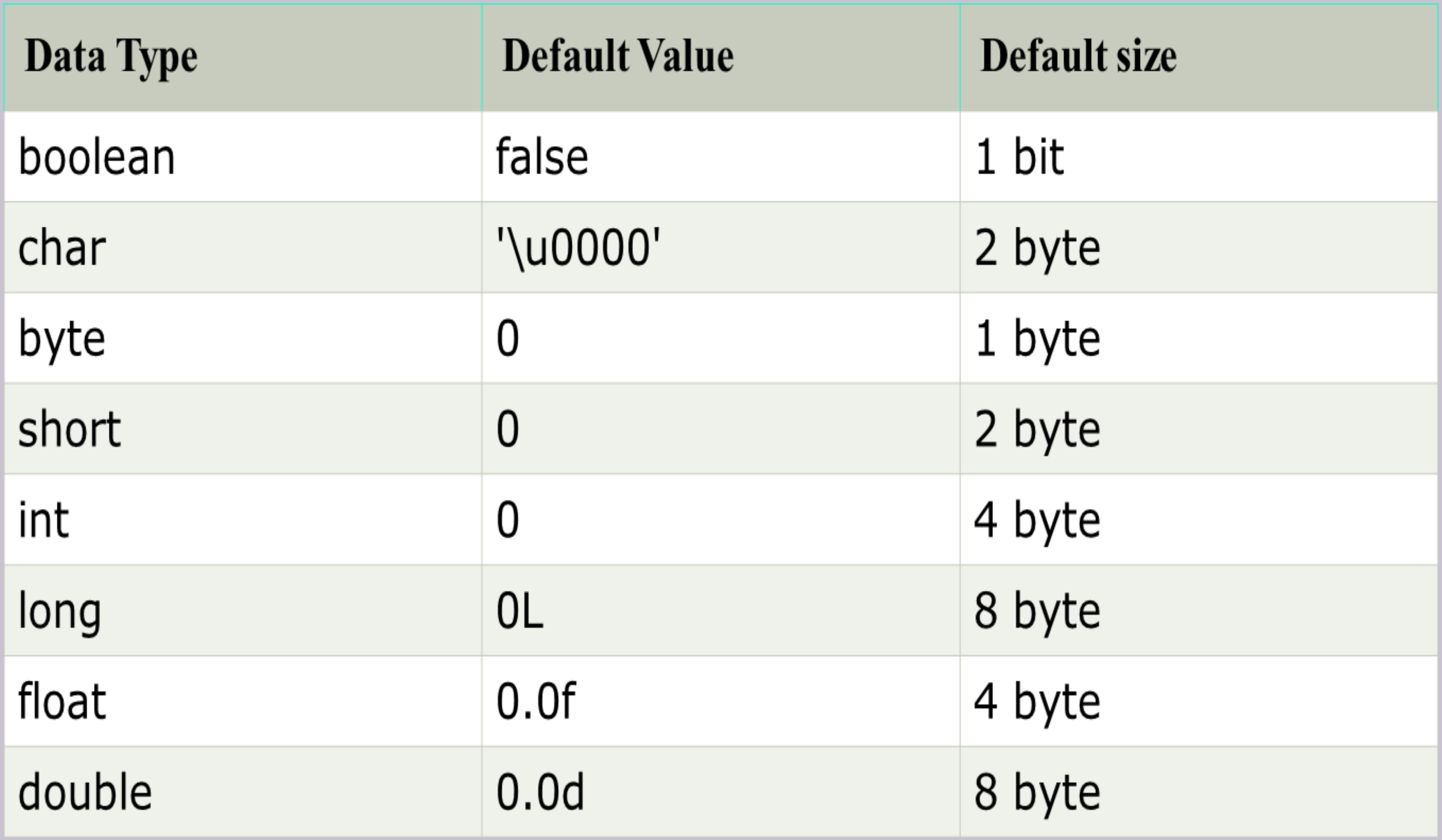
String literals: Always enclosed in double quotes

““,e.g. “Programming”, “Hello”

Data types in Java



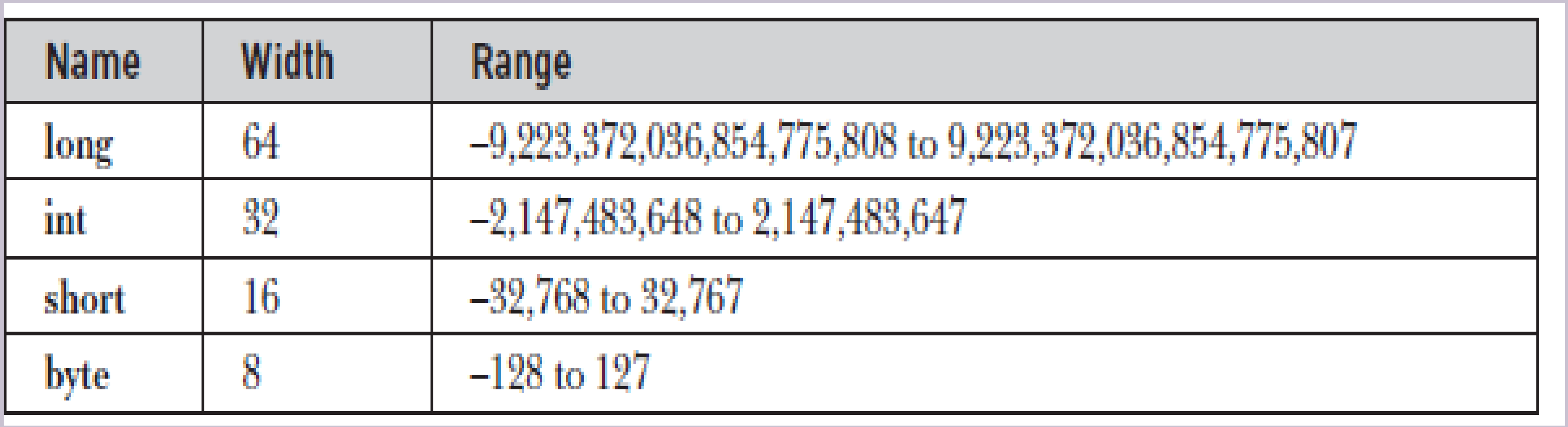
Storage size and default values



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

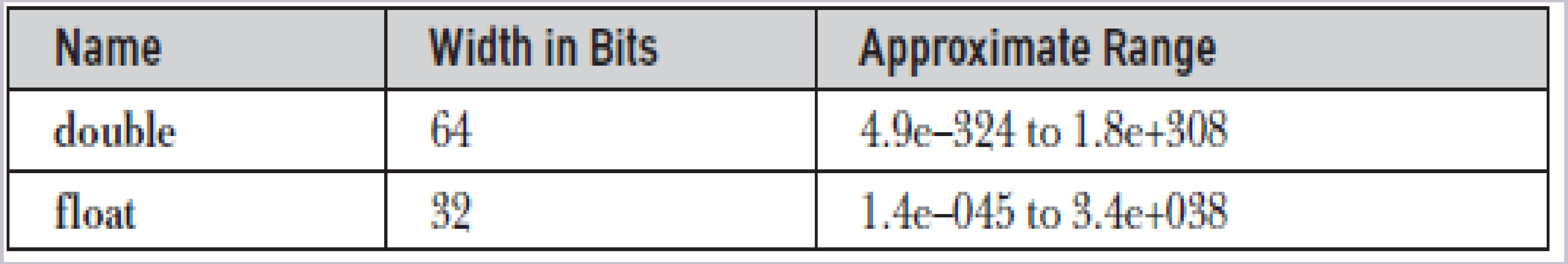
Ranges

Integers:



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

Floating-point types:



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

Ranges

Character:

- 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.
- At the time of Java's creation, Unicode required 16 bits. Thus, in Java char is a 16-bit type. The range of a char is 0 to 65,536.

Type conversion and Casting

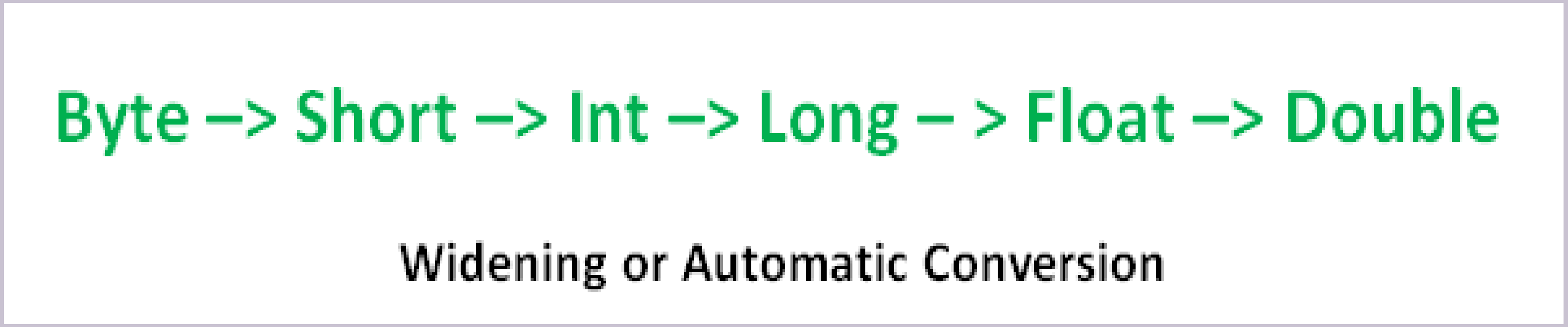
- ➢ It is fairly common to assign a value of one type to a variable of another type. If the two types are compatible, then Java will perform the conversion automatically.
- ➢ When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

- *The two types are compatible.*
- *The destination type is larger than the source type.*

- ➢ When these two conditions are met, a **widening** conversion takes place. For example, the **int** type is always large enough to hold all valid **byte** values, so no explicit cast statement is required.
- ➢ if we try to assign int typed value(larger ranged value) to byte(smaller ranged value), error would arise.[As here we are doing wrong assignment, so java compiler will not perform narrowing automatically]
- ➢ If we want to perform narrowing, we need to use explicit type casting

More points

In java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.



Byte –> Short –> Int –> Long – > Float –> Double

Widening or Automatic Conversion

Examples: Type conversion and casting

Example 1:

byte x=123;

int y=x;[Widening-Type conversion]//No error

Example 2:

int y=123;

byte x=y;[Error-Incompatible types]

Example 3:

int y=123;

byte x=(int)y;[Narrowing-Using cast expression]//No error

Automatic type promotions in java

Java defines several *type promotion* rules that apply to expressions. They are as follows:

- First, all **byte**, **short**, and **char** values are promoted to **int**, as just described.
- Then, if one operand is a **long**, the whole expression is promoted to **long**.
- If one operand is a **float**, the entire expression is promoted to **float**.
- If any of the operands are **double**, the result is **double**.

Example-Type promotion

```
public class Promote {
public static void main(String args[]) {
byte b = 42;
char c = 'a';
short s = 1024;
int i = 50000;
float f = 5.67f;
double d = .1234;
double result = (f * b) + (i / c) - (d * s);
System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
}
```

In the first subexpression, **f * b, b** is promoted to a **float** and the result of the subexpression is **float**. Next, in the subexpression **i/c, c** is promoted to **int**, and the result is of type **int**. Then, in **d * s**, the value of **s** is promoted to **double**, and the type of the subexpression is **double**. Finally, these three intermediate values, **float**, **int**, and **double**, are considered. The outcome of **float** plus an **int** is a **float**. Then the resultant **float** minus the last **double** is promoted to **double**, which is the type for the final result of the expression.

Writing Your First Java Program

```
class MyJavaProgram
    {
            public static void main(String args[])
                    {
                            System.out.println(“Have fun in Java…”);
                    }
    }
```

- Understanding first java program
- Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().
- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument.
- **System.out.println()** is used print statement.

- System.out.println is a Java statement that prints the argument passed, into the **System.out** which is generally stdout.
- System is a Class
- out is a Static Member Field
- println() is a method
- System is a class in the **java.lang package** . The out is a static member of the System class, and is an instance of **java.io.PrintStream** . The println is a method of java.io.PrintStream. This method is overloaded to print message to output destination, which is typically a console or file.
- class System {
-   public static final PrintStream out;
-   //...
- }

the System class belongs to java.lang package

```
class PrintStream{
  public void println();
  //...
}
```

the Prinstream class belongs to java.io package

Compiling and Executing Java Program

Step-1: Save your file with .java extension.

- Example: Program1.java

NOTE: If the class is public then the file name MUST BE same as the name of the class.

Step-2: Compile your .Java file using javac compiler from the location where the file is saved.

```
javac Program1.java
```

Compiling and Executing Java Program

Step-3: Execute your java class which contains the following method: public static void main(String args[]) {}

```
java MyJavaProgram
```

Q1

Which of the following is an invalid identifier?

A. $abc

B. _abc

C. abc_pqr

D. #abc

Q2

Which of the following is valid identifier name?

A. abc#pqr

B. 1abc

C. a$b

D. @abc

Q3

What will be the output of following code?

```
public class abc2
{
public static void main(String[] args)
{
long x=123;
int y=x;
System.out.println(y);
}
}
```

A. 123

B. 123000

C. Compile time error

D. 0

Q4

What will be the output of following code?

```
public class abc2
{
public static void main(String[] args)
{
double x=123.56;
int y=(int)x;
System.out.println(y);
}
}
```

A. 123.0
B. 123
C. Compile time error
D. Runtime error

Q5

What will be the output of following code?

```
public class abc2
{
public static void main(String[] args)
{
byte a=127;
a++;
System.out.println(a);
}
}
```

A. 128

B. -128

C. 0

D. Compile time error

Q6

Output?

```
public class abc2
{
public static void main(String[] args)
{
char x=65;
System.out.println(x);
}
}
```

A. A

B. 65

C. Garbage value

D. Some special character will be printed

What will be the output of following code?(Q7)

```
public class First
{
public static void main(String[] args)
{
int a=0xC;
int b=0b1111;
System.out.println(a+" "+b);
}
}
```

A. 12 14

B. 11 10

C. Error

D. 12 15

Q8

The smallest integer type is.........and its size is.......bits.

A.short, 8

B.byte, 8

C.short, 16

D.short, 16

Q9

In Java,the word *true* is

A. A Java keyword

B. A Boolean literal

C. Same as value 1

D. Same as value 0

Q10

Automatic type conversion in Java takes place when

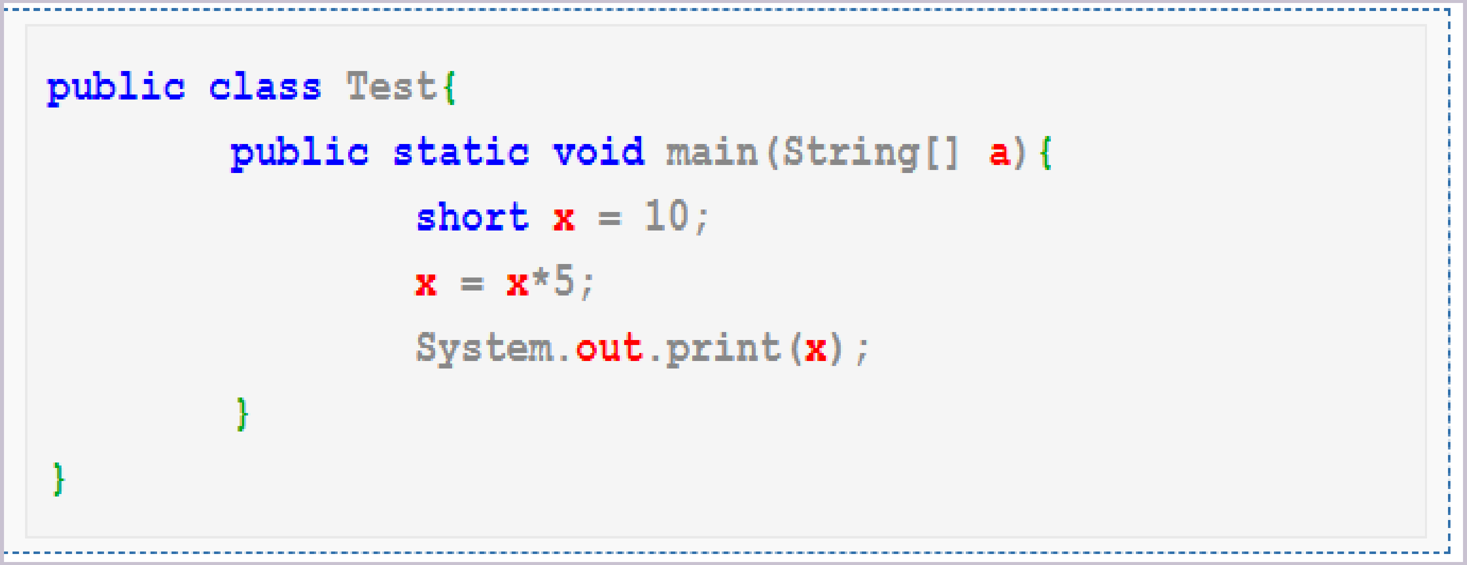
A. Two type are compatible and size of destination type is shorter than source type.

B. Two type are incompatible and size of destination type is equal of source type.

C. Two type are compatible and size of destination type is larger than source type.

D. All of the above

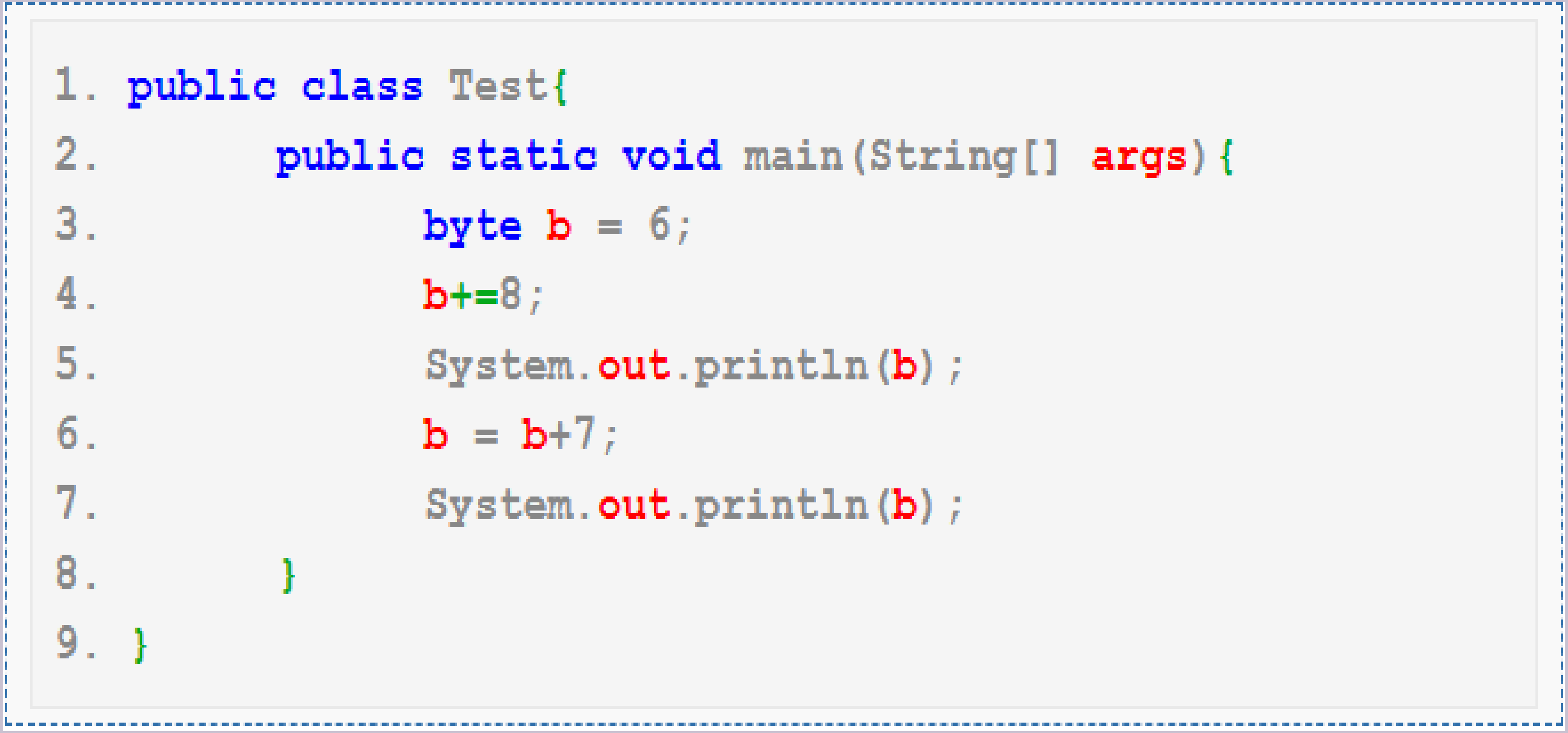
Q11



```
public class Test{
        public static void main(String[] a){
                short x = 10;
                x = x*5;
                System.out.print(x);
        }
}
```

A.50
B.10
C.Compilation Error
D.None of these

Q12



```
1. public class Test{
2.         public static void main(String[] args){
3.                 byte b = 6;
4.                 b+=8;
5.                 System.out.println(b);
6.                 b = b+7;
7.                 System.out.println(b);
8.         }
9. }
```

A. 14 21
B. 14 13
C. Compilation fails with an error at line 6
D. Compilation fails with an error at line 4

INTRODUCTION

Outcome of the Lecture

Upon completion of this lecture you will be able to understand

- ✓ Fundamentals and Characteristics of Java Language
- ✓ Basic Terminology – JVM, JRE, API, JDK, IDE, bytecode
- ✓ Writing, Compiling and Executing Java Program

Outline of the Presentation

- ✓ Overview of Java
- ✓ History of Java
- ✓ Java Standards
- ✓ Editions of Java
- ✓ JDK and IDE
- ✓ Characteristics of Java
- ✓ Sample Java Program
- ✓ Creating, Compiling and Executing Java Program
- ✓ Java Runtime Environment (JRE) and Java Virtual Machine (JVM)

Overview of Java

- A general-purpose Object-Oriented language
- Developed by Sun Microsystems, later acquired by Oracle Corporation
- Java can be used to develop
  - ✓ Stand alone applications
  - ✓ Web applications.
  - ✓ applications for hand-held devices such as cell phones

History of Java

- Developed by a team led by James Gosling at Sun Microsystems in 1991 for use in embedded chips .
- The primary motivation was the need for a platform-independent (that is, architecture-neutral) language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls.
- Originally called Oak.
- In 1995 redesigned for developing Internet applications and renamed as Java
- HotJava - The first Java-enabled Web browser (1995)

Java Standards

Computer languages have strict rules of usage. Java standards are defined by

- The Java language specification and
- Java API

The *Java language specification* is a technical definition of the language that includes the syntax and semantics of the Java programming language.

The *Application Program Interface (API)* contains predefined classes and interfaces for developing Java programs(or Java library)

Java language specification is stable, but the API is still expanding

Editions of Java

- ✓ Java Standard Edition (SE)

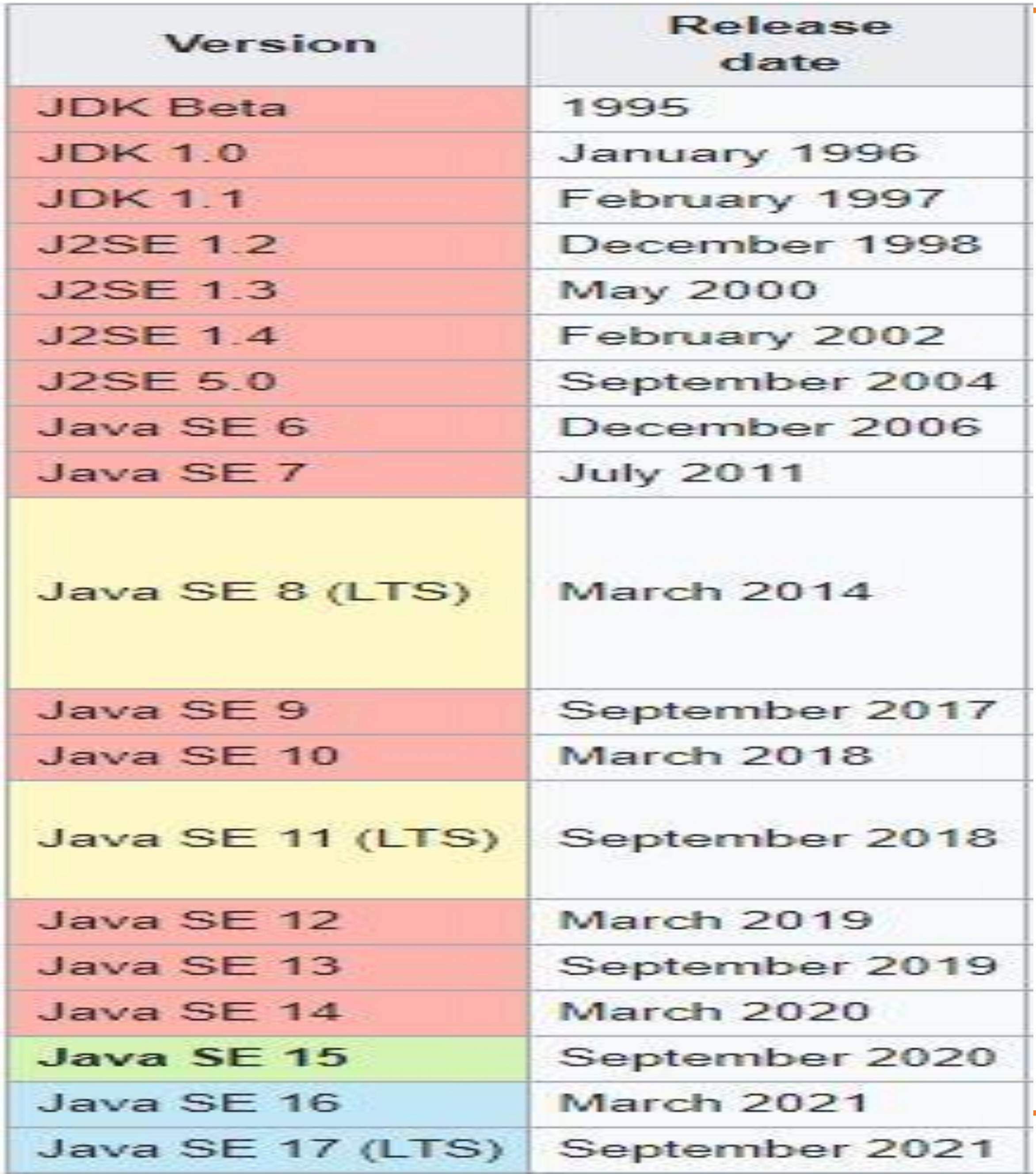
  used to develop client-side standalone applications or applets.
- ✓ Java Enterprise Edition (EE)

  used to develop server-side applications such as Java servlets and Java ServerPages.
- ✓ Java Micro Edition (ME).

  used to develop applications for mobile devices such as cell phones.

We will be using Java SE

Various versions of Java over the years



Version	Release date
JDK Beta	1995
JDK 1.0	January 1996
JDK 1.1	February 1997
J2SE 1.2	December 1998
J2SE 1.3	May 2000
J2SE 1.4	February 2002
J2SE 5.0	September 2004
Java SE 6	December 2006
Java SE 7	July 2011
Java SE 8 (LTS)	March 2014
Java SE 9	September 2017
Java SE 10	March 2018
Java SE 11 (LTS)	September 2018
Java SE 12	March 2019
Java SE 13	September 2019
Java SE 14	March 2020
Java SE 15	September 2020
Java SE 16	March 2021
Java SE 17 (LTS)	September 2021

Integrated Development Environment (IDE)

- ✓ Besides JDK, a Java development tool—software that provides an integrated development environment (IDE) for rapidly developing Java programs can be used.

- ✓ Editing, compiling, building, debugging, and online help are integrated in one graphical user interface.

  - Borland Jbuilder
  - Microsoft Visual J++
  - Net Beans
  - Eclipse
  - BlueJ

Characteristics of Java

- Java Is Simple
- Java Is Object-Oriented
- Java Is Secure
- Java Is Robust
- Java Is Interpreted
- Java Is Distributed
- Java Is Architecture-Neutral
- Java Is Portable
- Java's Performance
- Java Is Multithreaded
- Java Is Dynamic

Java Is Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Characteristics of Java

Java Is Object-Oriented

Java is inherently object-oriented. Object-oriented programming provides great flexibility, modularity, clarity, and reusability through encapsulation, inheritance, and polymorphism

Java Is Secure

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- No explicit pointer
- Java Programs run inside a virtual machine sandbox

Characteristics of Java

Java Is Robust

- Robust simply means strong. Java is robust because:
- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Java Is Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI's are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Characteristics of Java

Java Is Interpreted(While execution)

You need an interpreter to run Java programs. The programs are compiled into the Java Virtual Machine code called bytecode. The bytecode is machine-independent and can run on any machine that has a Java interpreter, which is part of the Java Virtual Machine (JVM).

Java Is Architecture-Neutral

Write once, run anywhere. With a Java Virtual Machine (JVM), you can write one program that will run on any platform.

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

•In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Java Is Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation

Characteristics of Java

Java's Performance

- Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.

Java provides Multithreading

- A thread allows multiple activities within a single process. 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 multi-media, Web applications, etc.

Java is Dynamic

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

JDK,JRE and JVM

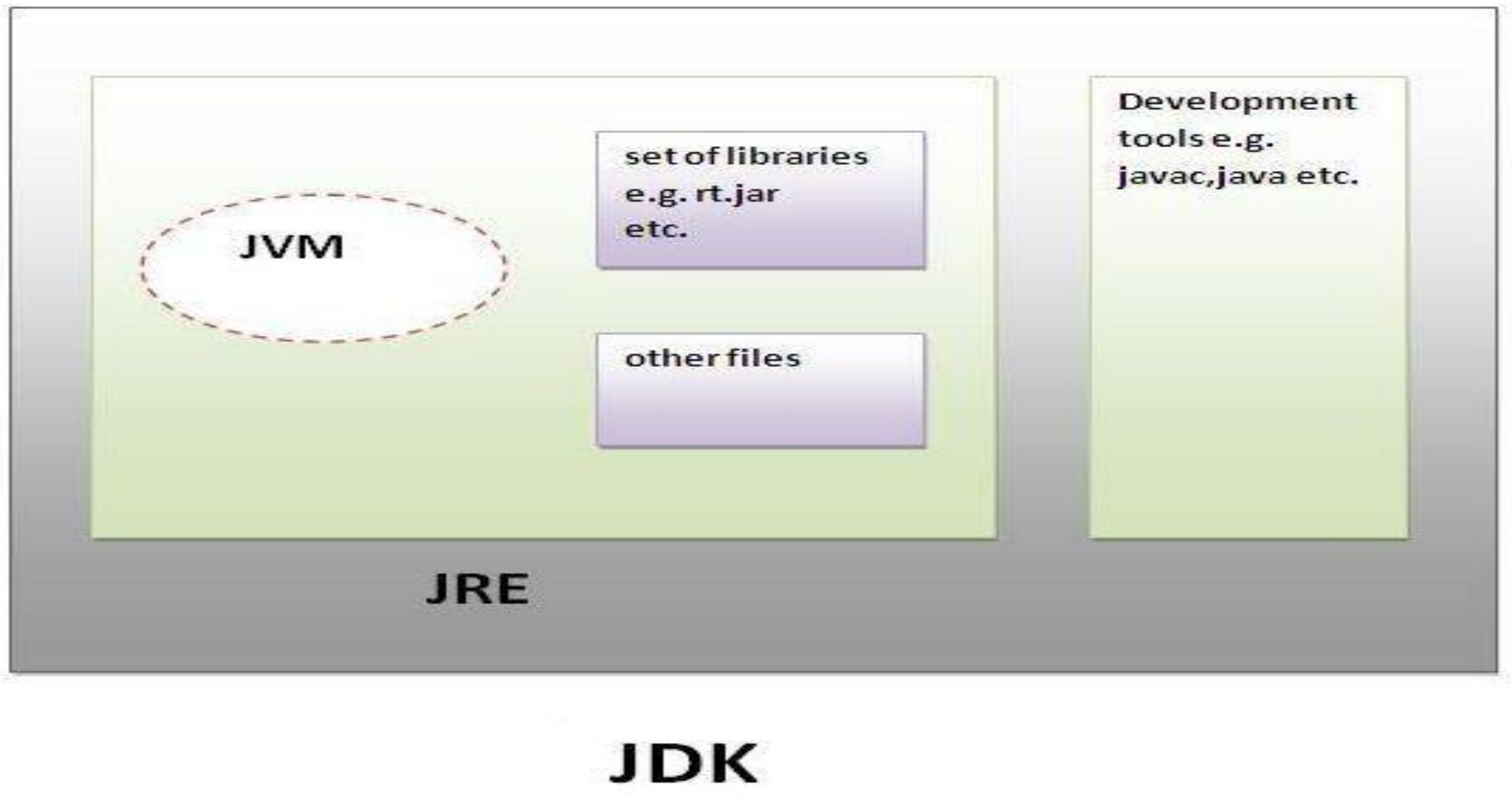
- **JDK – Java Development Kit** (in short JDK) is Kit which provides the environment to **develop and execute(run)** the Java program. JDK is a kit(or package) which includes two things
    - Development Tools(to provide an environment to develop your java programs)
    - JRE (to execute your java program).

  Note : JDK is only used by Java Developers.

- **JRE – Java Runtime Environment** (to say JRE) is an installation package which provides environment to **only run(not develop)** the java program(or application)onto your machine. JRE is only used by them who only wants to run the Java Programs i.e. end users of your system.

- **JVM – Java Virtual machine**(JVM) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for **executing the java program line by line** hence it is also known as interpreter.

Understanding JDK, JRE and JVM


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

Various components of JDK & JRE

JDK

- JDK is an acronym for *Java Development Kit.*
- It physically exists. It contains JRE and development tools.
- The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets.
- It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development.

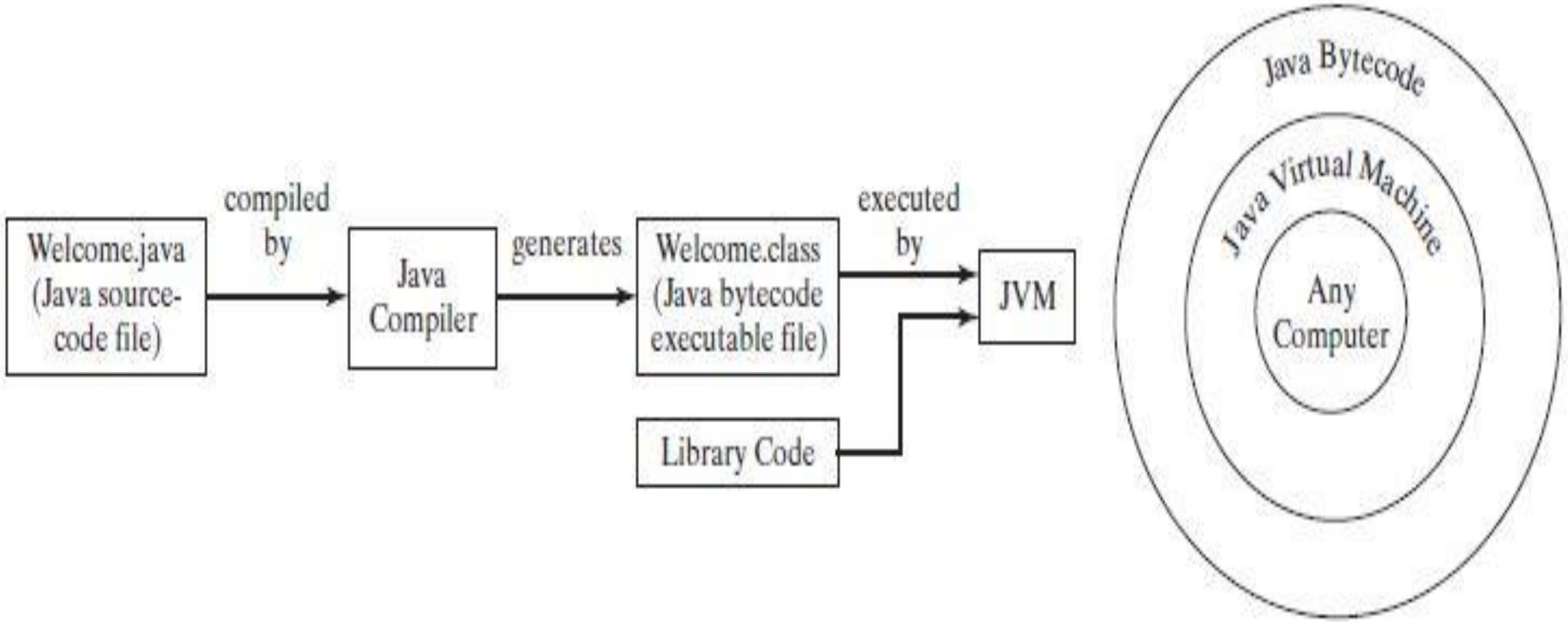
JRE

- JRE is an acronym for *Java Runtime Environment*.
- It is the implementation of JVM and used to provide runtime environment.
- It contains set of libraries and other files that JVM uses at runtime.

Understanding JVM

- ➢JVM (Java Virtual Machine) is an abstract machine.
- ➢It is a specification that provides runtime environment in which java byte code can be executed.
- ➢JVMs are available for many hardware and software platforms.

- ➢The JVM performs following main tasks:
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment


Welcome.java (Java source-code file)
compiled by
Java Compiler
generates
Welcome.class (Java bytecode executable file)
executed by
JVM
Library Code
Java Bytecode
Java Virtual Machine
Any Computer

Internal Architecture of JVM

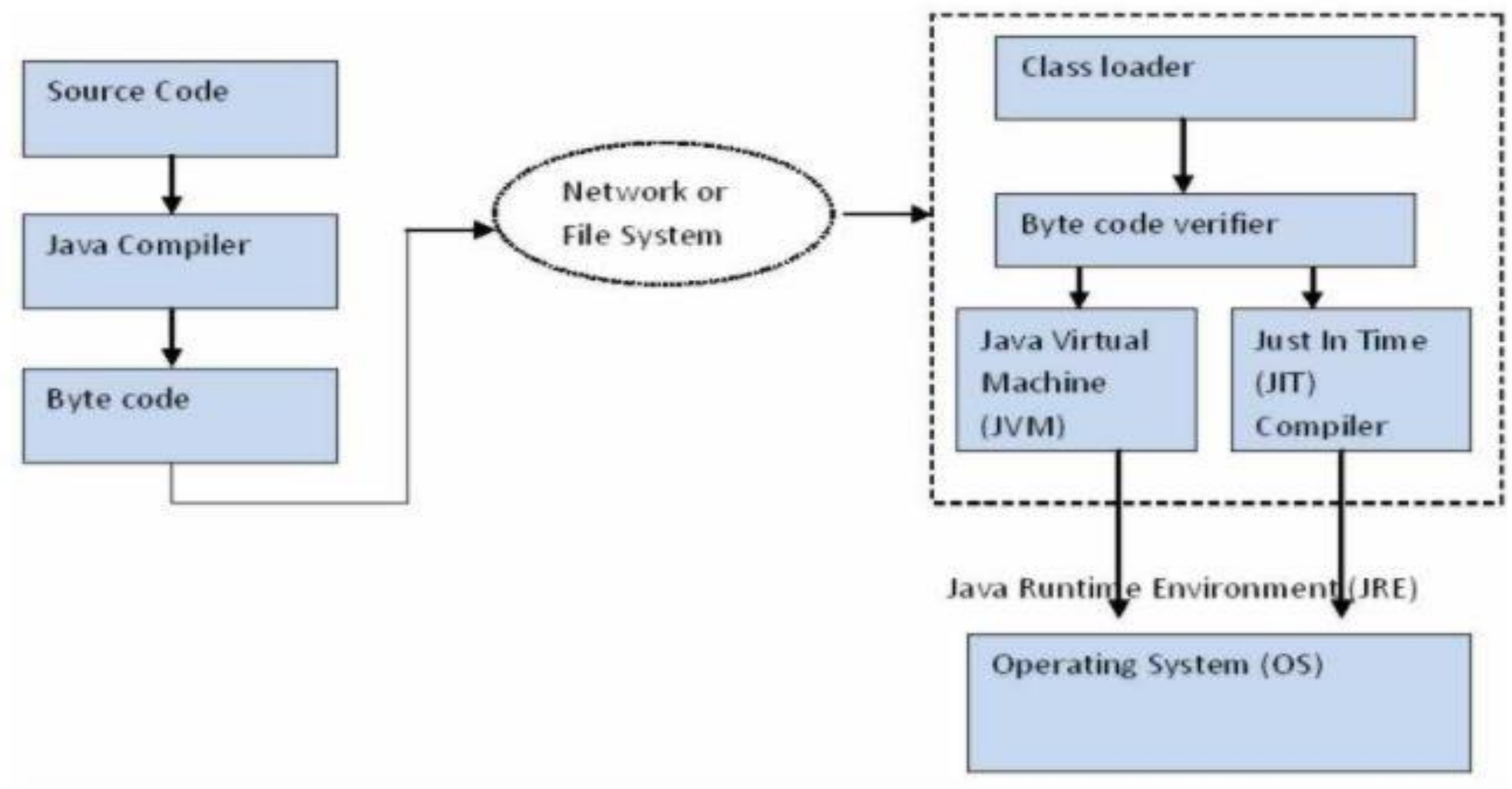
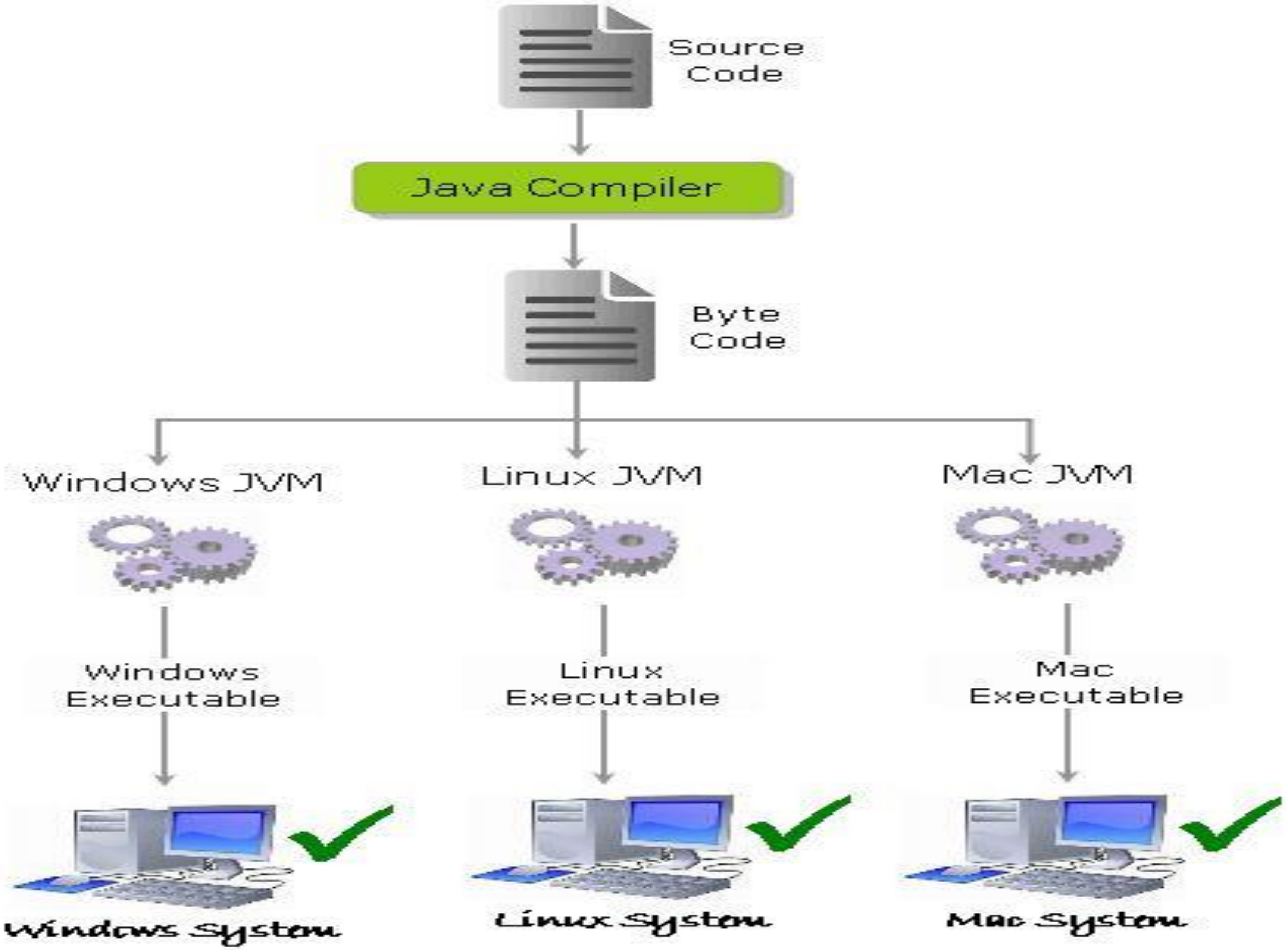


Figure 3: Java Architecture in Detail

How Java is Platform-independent?


Source Code
Java Compiler
Byte Code
Windows JVM
Linux JVM
Mac JVM
Windows Executable
Linux Executable
Mac Executable
Windows System
Linux System
Mac System

How Java is Platform-independent?

- The source code (program) written in java is saved as a file with .java extension.

- The java compiler "javac" compiles the source code and produces the platform independent intermediate code called BYTE CODE. It is a highly optimized set of instructions designed to be executed by the JVM.

How Java is Platform-independent?

- The byte code is not native to any platform because java compiler doesn't interact with the local platform while generating byte code.

- It means that the Byte code generated on Windows is same as the byte code generated on Linux for the same java code.

- The Byte code generated by the compiler would be saved as a file with .class extension. As it is not generated for any platform, can't be directly executed on any CPU.

Q1

Which of the following component generates the byte code?

A. javac

B. java

C. javadoc

D. jar

Q2

Java is

A. Compiled

B. Interpreted

C. Compiled as well as Interpreted

D. None of these

Q3

JVM is to generate

A. Bytecode
B. Native code
C. Source code
D. Machine independent code

Q4

What is the extension of bytecode file?

A. .exe

B. .java

C. .class

D. .obj

Java Libraries, Middle-ware, and Database options

- java.lang
- java.util
- java.sql
- java.io
- java.nio
- java.awt
- javax.swing

Sample Java Program

```
import java.lang.*; // Optional

public class Welcome
{
  public static void main(String[] args)
  {
    System.out.println("Welcome to Java!");
  }
}
```

Creating, Compiling and Executing Java Program

- Use any text editor or ~~IDE~~ to create and edit a Java source-code file

- The source file must end with the extension .java and must have exactly the same name as the public class name Welcome.java

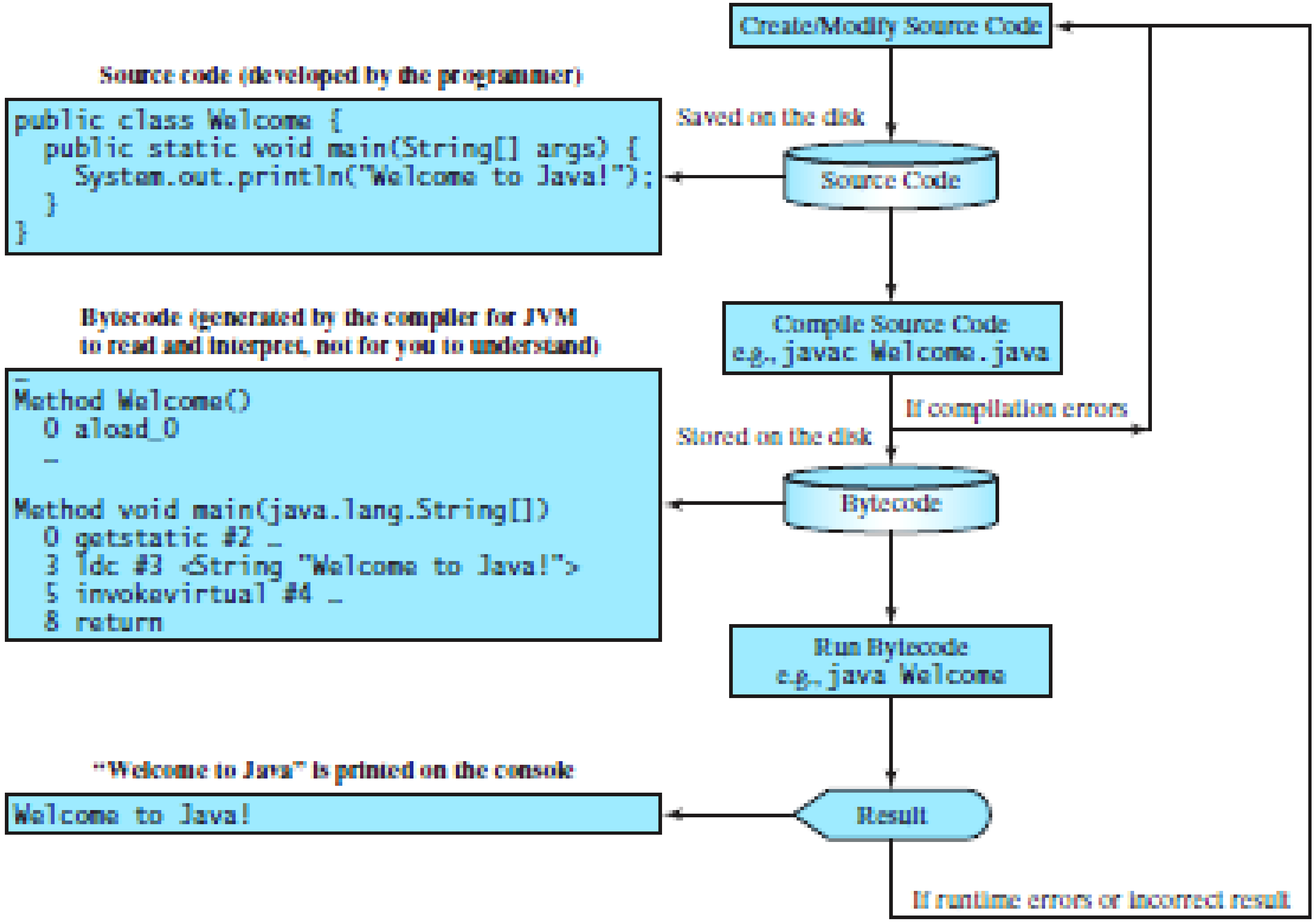
- Compile the .java file into java byte code file (Java bytecode is the instruction set of the Java virtual machine)

  javac Welcome.java

  If there are no syntax errors, the compiler generates a bytecode file with .class extension

- Run the byte code - java Welcome

Creating, Compiling and Executing Java Program


Create/Modify Source Code
Saved on the disk
Source Code
Compile Source Code
e.g., javac Welcome.java
If compilation errors
Stored on the disk
Bytecode
Run Bytecode
e.g., java Welcome
Result
If runtime errors or incorrect result
Source code (developed by the programmer)
Bytecode (generated by the compiler for JVM to read and interpret, not for you to understand)
"Welcome to Java" is printed on the console

```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

```
…
Method Welcome()
  0 aload_0
  …

Method void main(java.lang.String[])
  0 getstatic #2 …
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 …
  8 return
```

```
Welcome to Java!
```

CSE310: Programming in Java

Topic: Operators in Java

Outlines

- Introduction
- Assignment Operator
- Arithmetic Operator
- Relational Operator
- Bitwise Operator
- Conditional Operator
- Unary Operator

Introduction

- Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.

Assignment Operator

- One of the most common operators is the simple assignment operator "=".
- This operator assigns the value on its right to the operand on its left.

Example:

```
int salary = 25000;          double speed = 20.5;
```

Arithmetic Operators

- Java provides operators that perform addition, subtraction, multiplication, and division.

Operator	Description
+	**Additive operator (also used for String concatenation)**
-	**Subtraction operator**
*****	**Multiplication operator**
/	**Division operator**
%	**Remainder operator**

Example of arithmetic operators

```
// To show the working of arithmetic operators
class Example
    {
      public static void main(String args[])
      {
      int a=10;
      int b=5;
      System.out.println(a+b);//15
      System.out.println(a-b);//5
      System.out.println(a*b);//50
      System.out.println(a/b);//2
      System.out.println(a%b);//0
      }
    }
```

Compound Assignments

- Arithmetic operators are combined with the simple assignment operator to create compound assignments.
- Compound assignment operators are +=, -=, *=, /=, %=
- For example, x+=1; and x=x+1; both increment the value of x by 1.

Relational Operators

- Relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand.

- It always returns boolean value i.e true or false.

Relational Operators

Operator	Description
==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Example of relational operator

```
// To show the working of relational operators
class Example
      {
        public static void main(String args[])
        {
        int a=10;
        int b=5;
        System.out.println(a>b);//true
        System.out.println(a<b);//false
        System.out.println(a==b);//false
        System.out.println(a!=b);//true
        }
      }
```

Unary Operators

- The unary operators require only one operand.

Operator	Description
+	**Unary plus operator; indicates positive value**
-	**Unary minus operator; negates an expression**
++	**Increment operator; increments a value by 1**
--	**Decrement operator; decrements a value by 1**
!	**Logical complement operator; inverts the value of a boolean**

Examples

```
// To show the working of  ++ and -- operator
class Example
    {
      public static void main(String args[])
      {
      int x=10;
      System.out.println(x++);//10 (11)
      System.out.println(++x);//12
      System.out.println(x--);//12 (11)
      System.out.println(--x);//10
      }
    }
```

Boolean Logical Operators

- The Boolean logical operators shown here operate only on boolean operands.

Operator	**Result**
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT

- The following table shows the effect of each logical operation:

A	**B**	**A | B**	**A & B**	**A ^ B**	**! A**
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

Short-Circuit Logical Operators (&& and ||)

- These are secondary versions of the Boolean AND and OR operators, and are known as short-circuit logical operators.

- OR (||) operator results in true when A is true , no matter what B is. Similarly, AND (&&) operator results in false when A is false, no matter what B is.

Example of logical and short-circuited operators

```
// To show the working of logical and shortcircuited operators
class Example
    {
      public static void main(String args[])
      {
      int a=10;
      int b=5;
      int c=20;
      System.out.println(a>b||a++<c);// true
      System.out.println(a);//10 because second condition is not checked
      System.out.println(a>b|a++<c);//true | true = true
      System.out.println(a);//11 because second condition is checked
      }
    }
```

```
// To show the working of  short circuited && and Logical & operator
class Example
      {
        public static void main(String args[])
        {
        int a=10;
        int b=5;
        int c=20;
        System.out.println(a<b&&a++<c);//false
        System.out.println(a);//10 because second condition is not checked
        System.out.println(a<b&a++<c);//false & true = false
        System.out.println(a);//11 because second condition is checked
        }
      }
```

The ? Operator

- Java includes a special ternary (three-way)operator, ?, that can replace certain types of if-then-else statements.
- The ? has this general form:

  expression1 ? expression2 : expression3

- Here,expression1 can be any expression that evaluates to a boolean value.

- If expression1 is true , then expression2 is evaluated; otherwise, expression3 is evaluated.

- Both expression2 and expression3 are required to return the same type, which can't be void.

int ratio = denom == 0 ? 0 : num / denom **;**

- When Java evaluates this assignment expression, it first looks at the expression to the left of the question mark.
- If denom equals zero, then the expression between the question mark and the colon is evaluated and used as the value of the entire ? expression.
- If denom does not equal zero, then the expression after the colon is evaluated and used for the value of the entire ? expression.
- The result produced by the? operator is then assigned to ratio.

Bitwise Operators

These operators act upon the individual bits of their operands.

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

Example: Bitwise operators

```
// To show the working of  & | ^ operator
class Example
    {
     public static void main(String args[])
     {
     byte a=4; // 00000100
     byte b=5; // 00000101
     System.out.println(a&b);//(00000100)4
     System.out.println(a|b);//(00000101)5
     System.out.println(a^b);//(00000001)1
     }
    }
```

Representation of –ve number in java[2’s Complement form]

Example:

-10&-20

10(00001010)

Taking 2's complement:

```
11110101
+        1
-----------------
11110110(-10)
```

20(00010100)

Taking 2's Complement:

```
11101011
+        1
----------------
11101100(-20)
```

Taking Bitwise &

```
11110110
11101100
-----------------
11100100[Here MSB is 1 so answer will be -ve]
```

Taking 2's complement again to get the final result

```
00011011
+        1
-----------------
```

00011100(28)-->Final answer -28[As MSB was already observed to be 1, hence 28 will be represented as -28]

The Left Shift Operator

- The left shift operator,<<, shifts all of the bits in a value to the left a specified number of times.

value << num

- Example:

00000110	6<< 2
00011000	24

The Right Shift Operator

- The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times.
  value >> num
- It is also known as signed right shift.
- Example:

00001000	8>> 2
00000010	2

For positive numbers

```
// To show the working of  << and >> operator
class Example
    {
     public static void main(String args[])
     {
     byte x=10;
     System.out.println(x<<2);// 10*2^2=40
     System.out.println(x>>2);// 10/2^2=2
     }
    }
```

For negative numbers

```
// To show the working of  << and >> operator
class Example
   {
    public static void main(String args[])
    {
    byte x=-10;
    System.out.println(x<<2);// 10*2^2=-40
    System.out.println(x>>2);// 10/2^2 -1=-3[-1 will be added if not completely
   divisible,otherwise Number/2^no.of bits]
    }
   }
```

The Unsigned Right Shift

- In these cases, to shift a zero into the high-order bit no matter what its initial value was. This is known as an unsigned shift.

- To accomplish this, we will use Java’s unsigned, shift-right operator, >>>, which always shifts zeros into the high-order bit.

- Example:
  - 11111111 11111111 11111111 11111111 –1 in binary as an int
  - >>>24
  - 00000000 00000000 00000000 11111111 255 in binary as an int

Unsigned right shift example

Take example of -1(which will be represented as 2’s Complement of 1)

00000000 00000000 00000000 00000001(1)[32 bit representation]

Taking 2's Complement:

11111111 11111111 11111111 11111110

```
+                                                   1
------------------------------------------------------------
```

11111111 11111111 11111111 11111111(-1)

>>>24[Unsigned right shift][Shifting by 24 bits]

00000000 00000000 00000000 11111111(255)[Here higher order bits are replaced with 0[No matter what the sign was][Here no need to take 2’s complement again to get final answer, it will be 255]

But if we use:

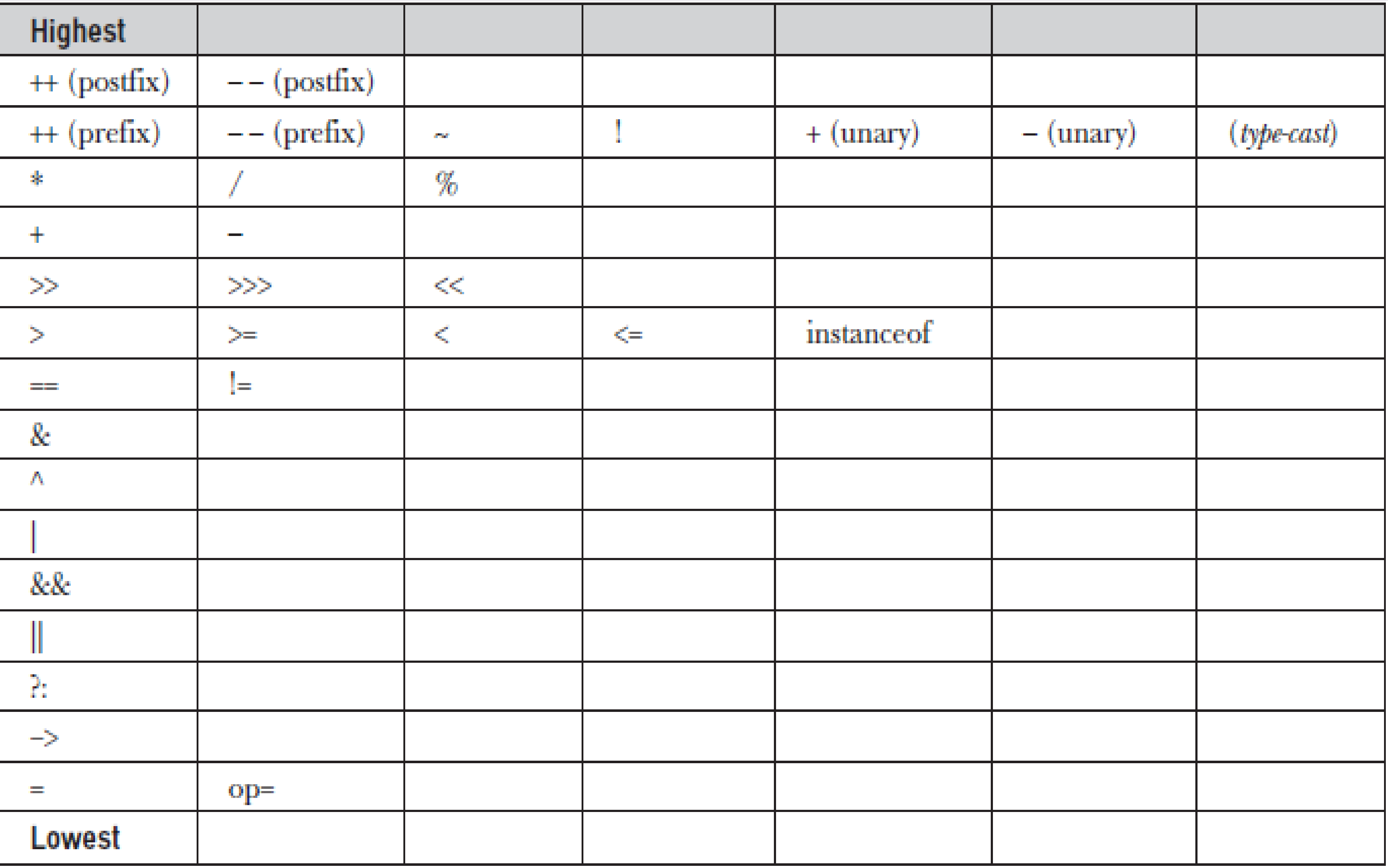
>>24

It will be:

11111111 11111111 11111111 11111111[Here we need to take 2’s complement

Taking 2’s Complement again, and it wil be -1

Operator Precedence



Highest						
++ (postfix)	– – (postfix)					
++ (prefix)	– – (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
–>						
=	op=					
Lowest						

Q1

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
System.out.println(20+2%3*5-10/5);
}
}
```

A. 5

B. 28

C. 10

D. 0

Q2

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
int a=6,b=3,c=2;
System.out.println(a>b+c++);
}
}
```

A. true

B. false

C. 6

D. 5

Q3

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
int a=100;
boolean b=false;
System.out.println(++a>100&!b);
}
}
```

A. true

B. false

C. 100

D. -1

Q4

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
int a=6,b=7;
boolean c;
c=++a==b||b++>=8;
System.out.println(c+" "+b);
}
}
```

A. true 8

B. false 7

C. true 7

D. false 8

Q5

What will be the output of the following code snippets?

```
public class First
{
public static void main(String[] args)
{
System.out.println(12^3);
}
}
```

A. 0

B. 15

C. 36

D. 9

Q6

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
System.out.print(2>1||4>3?false:true);
}
}
```

A. true

B. false

C. -1

D. Error

Q7

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
byte b=14;
System.out.println(b>>3);
}
}
```

A. 112

B. 1

C. 0

D. 17

Q8

What will be the output of followng code?

```
public class Test {
   public static void main(String args[])  {
      System.out.println(10  +  20 + "Hello");
      System.out.println("Hello" + 10 + 20);
   }
}
```

A. 30Hello

    Hello30

B. 1020Hello

    Hello1020

C. 30Hello

    Hello1020

D. 1020Hello

    Hello30

Q9

In Java, after executing the following code what are the values of x, y and z?

int x,y=10; z=12; x=y++ + z++;

A. x=22, y=10, z=12

B. x=24, y=10, z=12

C. x=24, y=11, z=13

D. x=22, y=11, z=13

CSE310: Programming in Java

Topic: Variables & their Scope

Outlines

- Variables and their Scope
  - Local Variable
  - Instance Variable
  - Class Variable

Variables

- Variable are the data members or the fields defined inside a class.

- There are three types of variables:
  - Local Variable
  - Instance Variable
  - Class Variable

Local Variable

- Local variables are those data members which are declared in methods, constructors, or blocks.
- They are created only when the method, constructor or block is executed and destroyed as soon as the execution of that block is completed.
- So the visibility of local variables is inside the block only, they can’t be accessed outside the scope of that block.

Important points…

- Access modifiers are NOT ALLOWED with local variables.

- Local variables are created in STACK.

- Default value is NOT ASSIGNED to the local variables in Java.
- Local variables must be given some value at the point of its declaration/or must be assigned some value before their usage

```
// Example of local variable
class Example
    {
     public static void main(String args[])
     {

          int x=12;// local to main method
          System.out.println(x);
     }
    }
```

Instance Variables

- Variables which are declared in a class, but outside a method, constructor or any block are known as instance variables.
- They are created inside the object in heap.
- Each object has its own set of instance variables and they can be accessed/modified separately using object reference.

- Instance variables are created when an object is created with the use of the 'new' keyword and destroyed when the object is destroyed.

- Access modifiers can be used with instance variables.

- Instance variables have default values.

- For numbers the default value is 0,
- For floating point variables, the default value is 0.0
- for Booleans it is false
- and for object references it is null.

```
// Example of instance variable
class Example
    {
     int x=0;// Instance Variable
     public static void main(String args[])
     {

          Example ob = new Example();
          System.out.println(ob.x);
     }
    }
```

Class Variable

- Class variables are also known as static variables.
- Variable which are declared in a class, but outside a method, constructor or a block and qualified with ‘static’ keyword are known as class variables.

- Only one copy of each class variable is created, regardless of how many objects are created from it.

- Static variables can be accessed by calling with the class name.
  ClassName.VariableName

- Static variables are created with the start of execution of a program and destroyed when the program terminates.
- Default values are same as instance variables.
- A public static final variable behaves as a CONSTANT in Java.
- They are stored in special heap area which is known as PermGen(Permanent Generation)-In Java 5 and 6
- In Java 8 PermGen is replaced with new terminology, known as “MetaSpace”
- The main reason for this change of PermGen in Java 8.0 is that it is tough to predict the size of PermGen, and it helps in improving garbage collection performance.

Example of static class variable

```
class Example
	{
	static int count;
		int return_count()
		{
			count++;
			return count;
		}
/*	If we do not use the static keyword then output will be
1
1
1*/
```

```
public static void main(String args[])
	{
	Example ob1= new Example();
	Example ob2= new Example();
	Example ob3= new Example();
System.out.println(ob1.return_count());
System.out.println(ob2.return_count());
System.out.println(ob3.return_count());
	}
}
```

Output:

```
1
2
3
```

Second example of static variables

```
class Example
        {
                String name;
        final static String section ="K19DUMMY";
                public static void main(String args[])
                {
                        Example ob1= new Example();
                        Example ob2= new Example();
                        Example ob3= new Example();
                        ob1.name="rohan";
                        ob2.name="mohan";
                        ob3.name="Ram";
        System.out.println(ob1.name+ob1.section);
        System.out.println(ob2.name+ob2.section);
        System.out.println(ob3.name+ob3.section);
}
}
```

We can fix the value of any member of the class which is common to all the objects of the class.

Variable Initialization

Local variables must be initialized explicitly by the programmer as the default values are not assigned to them where as the instance variables and static variables are assigned default values if they are not assigned values at the time of declaration.

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Memory allocation in java(2 types: Stack and Heap memory)
Stack memory

Stack Memory in Java is used for static memory allocation and the execution of a thread. It contains primitive values that are specific to a method and references to objects that are in a heap, referred from the method.

Access to this memory is in Last-In-First-Out (LIFO) order. Whenever a new method is called, a new block on top of the stack is created which contains values specific to that method, like primitive variables and references to objects.

When the method finishes execution, it’s corresponding stack frame is flushed, the flow goes back to the calling method and space becomes available for the next method.

Stack Memory

- It grows and shrinks as new methods are called and returned respectively
- Variables inside stack exist only as long as the method that created them is running
- It's automatically allocated and deallocated when method finishes execution
- If this memory is full, Java throws *java.lang.StackOverFlowError*
- Access to this memory is fast when compared to heap memory

Heap memory

Heap space in Java is used for dynamic memory allocation for Java objects and JRE classes at the runtime. New objects are always created in heap space and the references to this objects are stored in stack memory.

These objects have global access and can be accessed from anywhere in the application.

This memory model is further broken into smaller parts called generations, these are:

Young Generation – this is where all new objects are allocated and aged. A minor Garbage collection occurs when this fills up

Old or Tenured Generation – this is where long surviving objects are stored. When objects are stored in the Young Generation, a threshold for the object's age is set and when that threshold is reached, the object is moved to the old generation

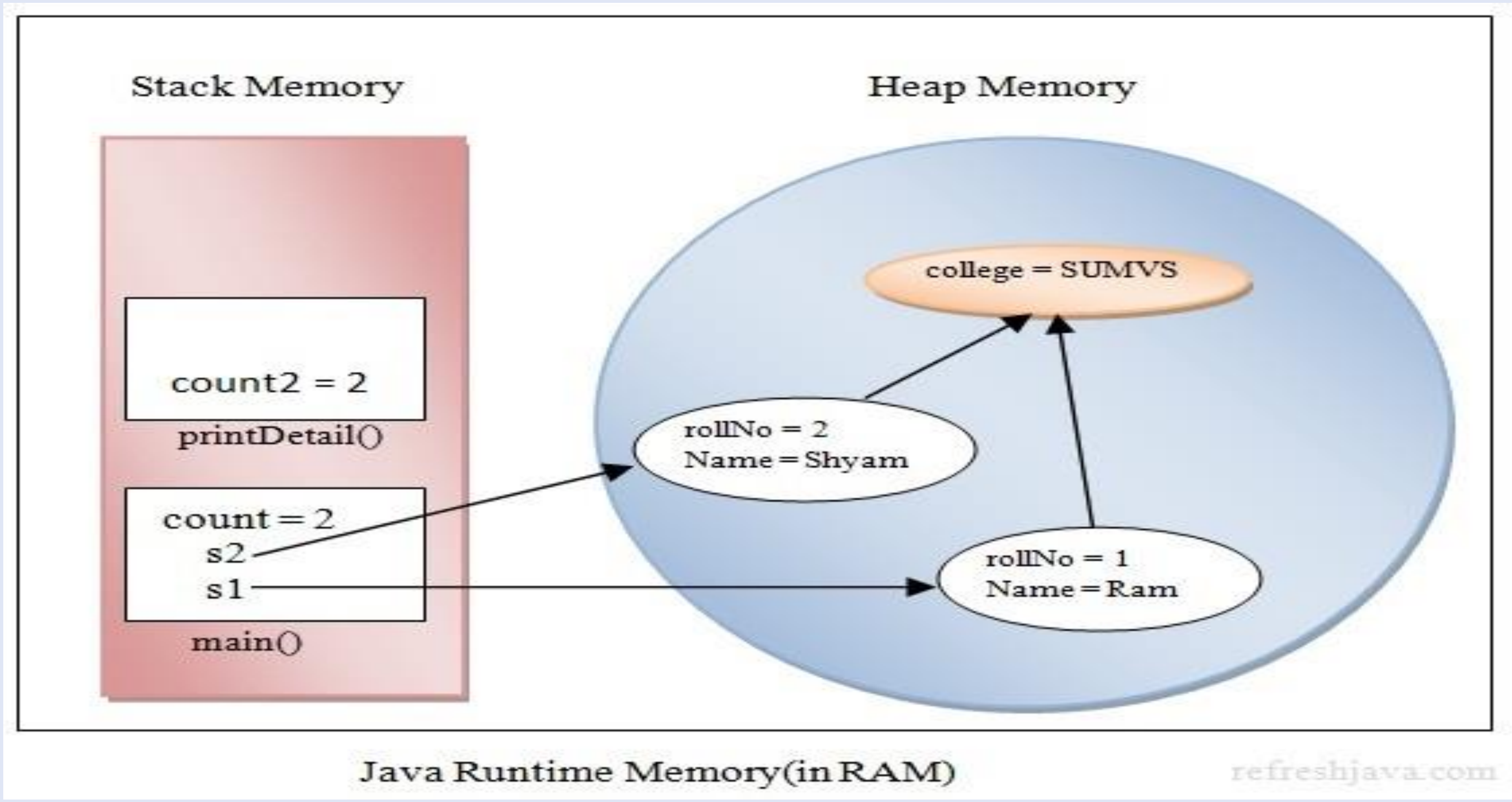
Permanent Generation – this consists of JVM metadata for the runtime classes and application methods, static or class variables also.

Key features of Heap memory

- It's accessed via complex memory management techniques that include Young Generation, Old or Tenured Generation, and Permanent Generation
- If heap space is full, Java throws *java.lang.OutOfMemoryError*
- Access to this memory is relatively slower than stack memory
- This memory, in contrast to stack, isn't automatically deallocated. It needs Garbage Collector to free up unused objects so as to keep the efficiency of the memory usage

Memory layout

Here rollNo,Name are object specific attributes(instance variables),college is the static member(or class variable),s1 and s2 are references to the objects,printDetail() is the method,count and count2 are the local variables of their respective methods



Q1

What will be the output of the following Program?

```
class VariableDemo
{
   public static void main(String [] args)
     {
            public int x = 10;
           System.out.print(x);
     }
}
```

A. 10
B. 0
C. Error
D. 1

Q2

What will be the output of the following Program?

```
class VariableDemo
{
   static int x=10;
   public static void main(String [] args)
     {
             int x;
            System.out.print(x);
     }
}
```

A. 0

B. Error

C. 1

D. 10

Q3

What will be the output of the following Program?

```
class VariableDemo
{
   static int x=5;
   public static void main(String [] args)
    {
             int x;
            System.out.print(VariableDemo.x);
    }
}
```

A. 5
B. 0
C. 1
D. Error

Q4

```
public class Main
{
    static int x;
    void increment()
    {
        System.out.print(x);
        x--;
    }
        public static void main(String[] args)
{
                Main obj1=new Main();
                Main obj2=new Main();
                obj1.increment();
                obj2.increment();
        }
}
```

A. 0 -1

B. 0 0

C. 1 1

D. -1 -1

Q5

```
public class Main
{
     int x;
    void increment()
    {
        System.out.print(x+" ");
        x=x+2;
    }
        public static void main(String[] args)
{
                Main obj1=new Main();
                Main obj2=new Main();
                obj1.increment();
                obj2.increment();
        }
}
```

A. 2 2

B. 0 2

C. 1 3

D. 0 0

?