Marwadi University

Object Oriented Programming

01CE0307

4 Credits

MU - CE – 3rd Sem

Prepared By

Prof. Ravikumar R Natarajan

Assistant Professor, CE Dept.

Marwadi University

Unit – 1
Introduction to Java

Contents

Marwadi University

History of Java

- Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need to recompile.
- **B led to C, C evolved into C++, and C++ set the stage for Java.**
- **James Gosling**, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The language was initially called **Oak** after an oak tree that stood outside Gosling's office.

Introduction

- Java syntax is defined in the **Java language specification**, and the Java library is defined in the Java **application program interface (API).** The **JDK (Java Development Kit)** is the software for compiling and running Java programs. **An IDE (Integrated Development Environment)** is for rapidly developing programs.
- Computer languages have strict rules of usage. If you do not follow the rules when writing a program, the computer will not be able to understand it. The Java language specification and the Java API define the Java standards.

Language Specification

- The Java language specification is a technical definition of the Java programming language's syntax and semantics. You can find the complete Java language specification at https://docs.oracle.com/javase/specs/ (Accessed on 21-Jan-22)

API

- The application program interface (API), also known as library, contains predefined classes and interfaces for developing Java programs.
- The API is still expanding. You can view and download the latest version of the Java API at https://www.oracle.com/in/java/technologies/javase-jdk8-doc-downloads.html (Accessed on 21-Jan-22)

Java Editions

- Java is a full-fledged and powerful language that can be used in many ways. It comes in three editions:
- Java **Standard Edition (Java SE)** to develop client-side applications. the applications can run on desktop.
- Java **Enterprise Edition (Java EE)** to develop server-side applications, such as Java servlets, **JavaServer Pages (JSP), and JavaServer Faces (JSF).**
- Java **Micro Edition (Java ME)** to develop applications for mobile devices, such as cell phones.

JDK & JRE

- The JDK consists of a set of separate programs, each invoked from a command line, for compiling, running, and testing Java programs.
- The program for running Java programs is known as JRE.
- Instead of using the JDK, you can use a Java development tool (e.g., NetBeans, Eclipse, and TextPad)—software that provides an integrated development environment (IDE) for developing Java programs quickly.
- Editing, compiling, building, debugging, and online help are integrated in one graphical user interface (GUI). You simply enter source code in one window or open an existing file in a window, and then click a button or menu item or press a function key to compile and run the program.

Questions

- What is the Java language specification?
- What does JDK stand for? What does JRE stand for?
- What does IDE stand for?
- Are tools like NetBeans and Eclipse are different languages from Java, or are they dialects or extensions of Java?

First Java Program

```
// Your First Program HelloWorld.java

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

Output

```
Hello, World!
```

Note: You need to give the name of the class and file name same in Java.

How Java "Hello, World!" Program Works?

1. // Your First Program

In Java, any line starting with// is a comment. Comments are intended for users reading the code to better understand the intent and functionality of the program. It is completely ignored by the Java compiler.

2. class HelloWorld { ... }

In Java, every application begins with a class definition. In the program, HelloWorld is the name of the class, and the class definition is:

```
class HelloWorld {
... .. ...
}
```

How Java "Hello, World!" Program Works?

3. public static void main(String[] args) { ... }

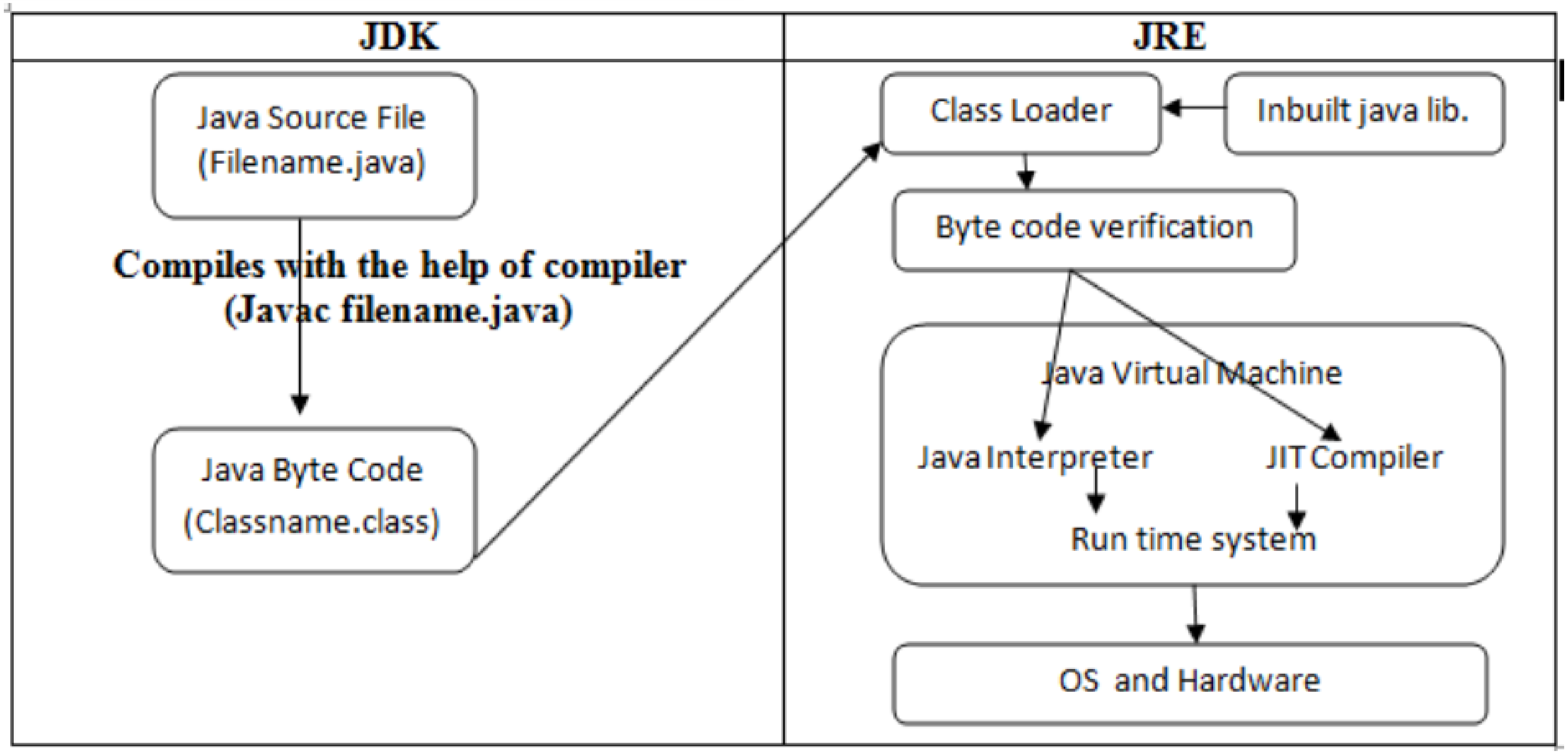
This is the main method. Every application in Java must contain the main method.
The Java compiler starts executing the code from the main method.

4. System.out.println("Hello, World!");

The code above is a print statement. It prints the text Hello, World! to standard
output (your screen). The text inside the quotation marks is called String in Java.

Notice the print statement is inside the main function, which is inside the class definition.

Compiling & Executing Java Program



Compiling & Executing Java Program

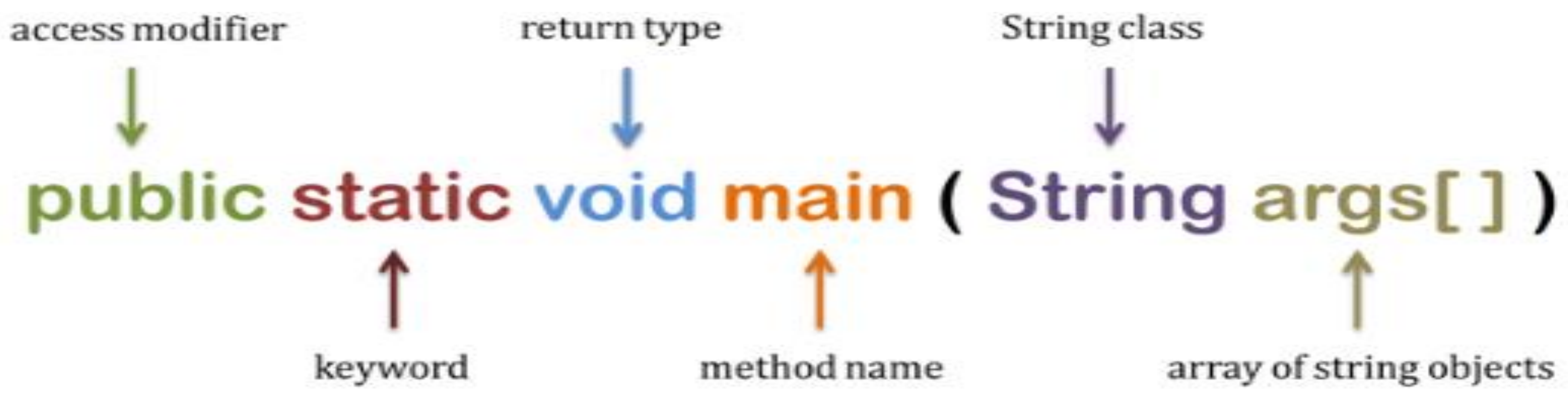
Java Program can be compiled using below command:

```
javac filename.java
```

And can be executed using below mentioned command:

```
java file-name
```

Main method in Java



Main method in Java

Every word in the public static void main statement has got a meaning to the JVM.

1. Public: It is an Access modifier, which specifies from where and who can access the method. Making the main() method public makes it globally available. It is made public so that JVM can invoke it from outside the class as it is not present in the current class.

2. Static: It is a keyword which is when associated with a method, makes it a class related method. The main() method is static so that JVM can invoke it without instantiating the class. This also saves the unnecessary wastage of memory which would have been used by the object declared only for calling the main() method by the JVM.

Main method in Java

3. Void: It is a keyword and used to specify that a method doesn't return anything. As main() method doesn't return anything, its return type is void. As soon as the main() method terminates, the java program terminates too. Hence, it doesn't make any sense to return from main() method as JVM can't do anything with the return value of it.

4. main: It is the name of Java main method. It is the identifier that the JVM looks for as the starting point of the java program. It's not a keyword.

5. String[] args: It stores Java command line arguments and is an array of type java.lang.String class. Here, the name of the String array is args but it is not fixed and user can use any name in place of it.

Questions

- Which file consists of Byte codes?
- What is JVM?
- What is JIT?
- **Is java a compiled or interpreted language?**
- Are tools like NetBeans and Eclipse are different languages from Java, or are they dialects or extensions of Java?

Objected Oriented Principles

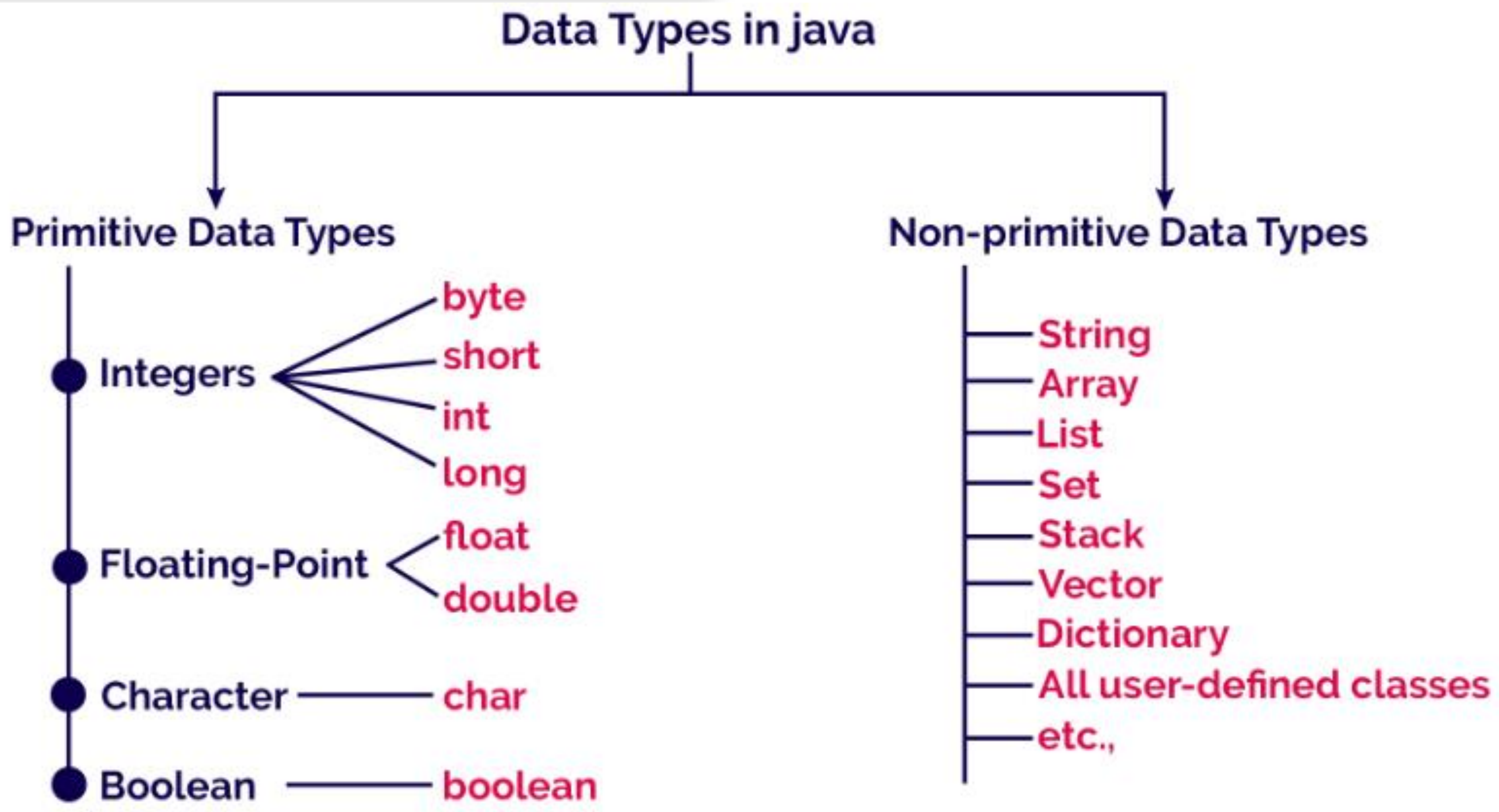
1. **Object** – Instance of class | - Run time entities which occupies memory
2. **Classes** – Collection of attributes, methods.
3. **Instance** – Obj. created at run time
4. **Inheritance** – Provides reusability |
5. **Data abstraction** – Information hiding | refers to particular feature and hiding its background details | used in s/w design phase.
6. **Encapsulation** – Binding data and method together | used in s/w implementation | Inherited
7. **Polymorphism** –Ability to take more than one form | Types: compile time & run time
8. **Message passing** – An object sends data to another obj.

Java Buzz Words

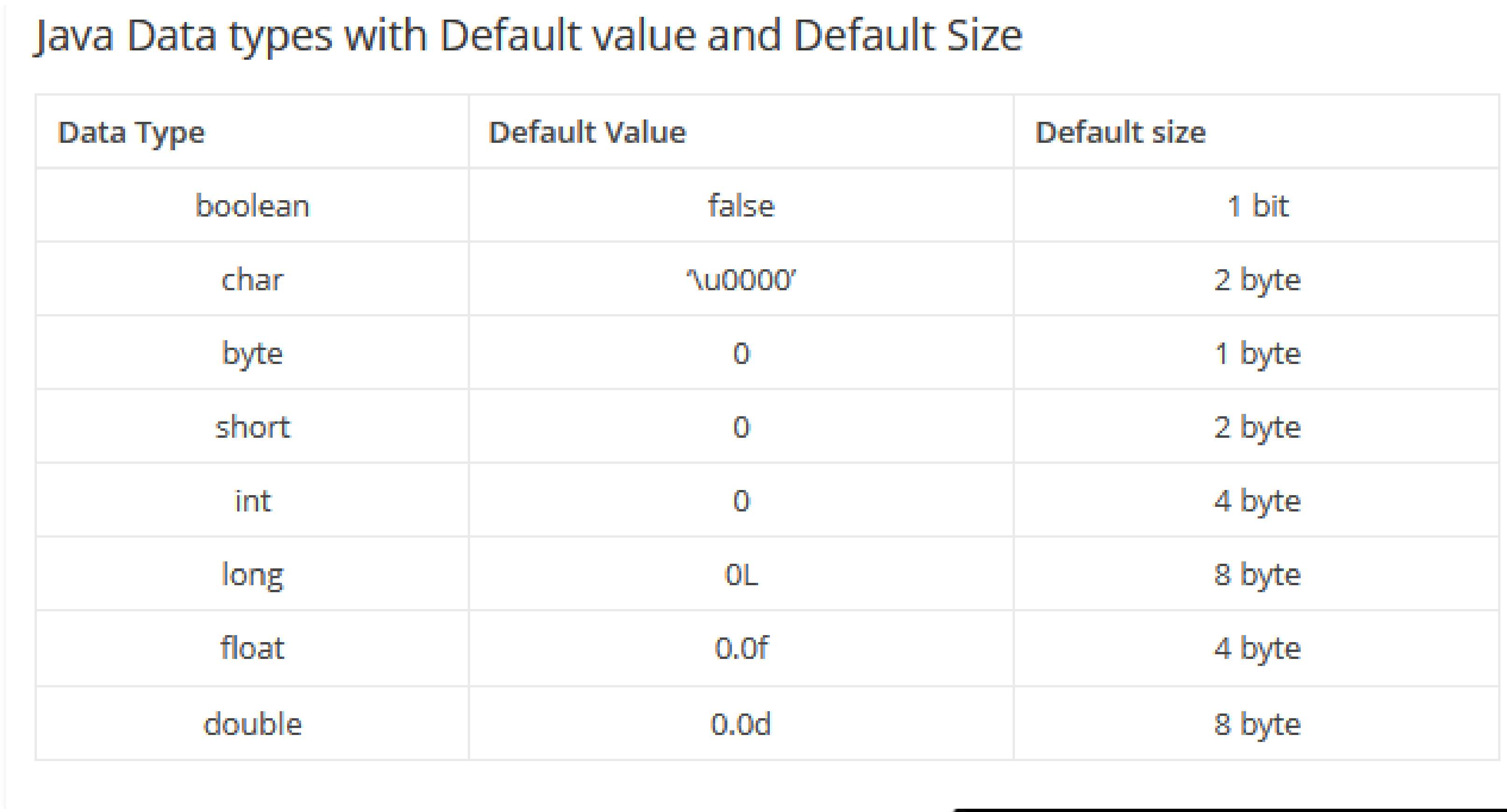
Marwadi University

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

Java Data types



Java Data types



Java Data types with Default value and Default Size

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

Java Data types

Character Data Type

A character data type represents a single character.

Unicode and ASCII code

Java supports Unicode, an encoding scheme to support the interchange, processing, and display of written texts in the world's diverse languages.

65,536 characters possible in a 16-bit encoding are not sufficient to represent all the characters in the world.

The Unicode standard therefore has been extended to allow up to 1,112,064 characters. Those characters that go beyond the original 16- bit limit are called supplementary characters.

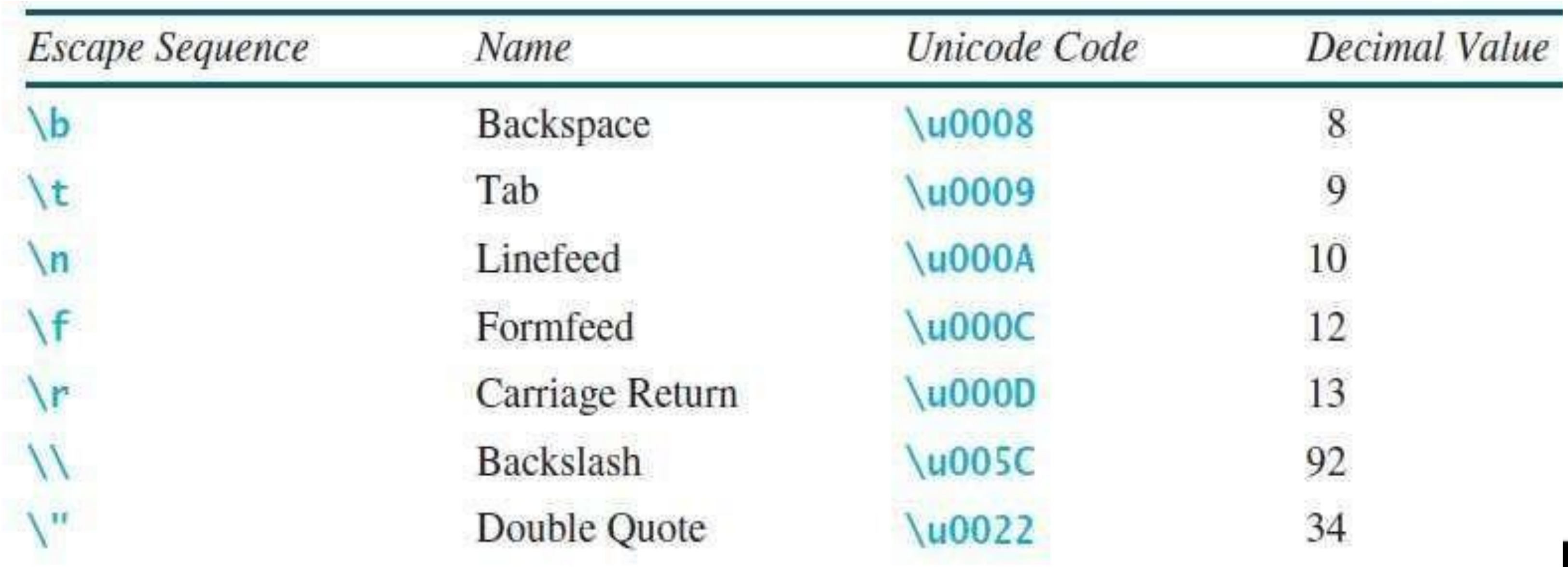
Escape Sequence for special characters

```
System.out.println("He said \"Java is fun\"");
```

output: **He said "Java is fun"**

```
System.out.println("\\t is a tab character");
```

output: **\t is a tab character**



Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed	\u000A	10
\f	Formfeed	\u000C	12
\r	Carriage Return	\u000D	13
\\	Backslash	\u005C	92
\"	Double Quote	\u0022	34

Java Identifiers

Identifiers are the names of variables, methods, classes, packages and interfaces.

Unlike literals they are not the things themselves, just ways of referring to them.

Literals are the values that are assigned to Identifiers. Ex: int count = 0;
In the HelloWorld program, HelloWorld, String, args, main and println are identifiers. The general rules for constructing names (naming conventions) for variables are:

› Names can contain letters, digits, underscores, and dollar signs

› Names must begin with a letter and it cannot contain whitespace

› Names can also begin with $ and _

› Names are case sensitive ("myVar" and "myvar" are different variables)

› Reserved words cannot be used as names

Variables in Java

Marwadi University

› Variables are containers for storing data values.

int - stores integers (whole numbers), without decimals, such as 123 or -123
float - stores floating point numbers, with decimals, such as 12.34 or -12.34
char - stores single characters, such as 'a' or ‘A'.
boolean - stores values with two states: true or false
String - stores text, such as "Hello".

› Syntax: Data-type variable-name = variable-value;

› Example: String name = "John";

Final Variables

You can add the final keyword if you don't want others (or yourself) to overwrite existing values.

› This will declare the variable as "final" or "constant", which means unchangeable and read-only.

› Example:

final float interest_rate = 7.85;
interst_rate = 8.23;   /* will generate an error: cannot assign a value to a final variable */

Java Naming conventions

- Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.

- But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.

- All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.

Java Naming conventions

By using standard Java naming conventions, you make your code easier to read for yourself and other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

Class

- It should start with the uppercase letter.
- It should be a noun such as Color, Button, System, Thread, etc. Use appropriate words, instead of acronyms.

```
public class Employee
{
//code snippet
}
```

Interface

- It should start with the uppercase letter.
- It should be an adjective such as Runnable, Remote, ActionListener. Use appropriate words, instead of acronyms.

```
interface Printable
{
//code snippet
}
```

Java Naming conventions

Method

- It should start with lowercase letter.
- It should be a verb such as main(), print(), println().
- If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerform

```
class Employee
{
// method
void draw()
{
//code snippet
}
}
```

Variable

- It should start with a lowercase letter such as id, name.
- It should not start with the special characters like & (ampersand), $ (dollar), _ (underscore). If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
- Avoid using one-character variables such as x, y, z.

```
class Employee
{
// variable
int id;
//code snippet
}
```

Java Naming conventions

Package

- It should be a lowercase letter such as java, lang.
- If the name contains multiple words, it should be

separated by dots (.) such as java.util, java.lang.

```
//package
package javaAppln01
class Employee
{
//code snippet
}
```

Constant

- It should be in uppercase letters such as RED, YELLOW.
- If the name contains multiple words, it should be
- separated by an underscore(_) such as MAX_PRIORITY.
- It may contain digits but not as the first letter.

```
class Employee
{
//constant
static final
int MIN_AGE = 18;
//code snippet
}
```

Java Naming conventions

- Using the right letter case is the key to following a naming convention:
- Lowercase is where all the letters in a word are written without any capitalization (e.g., while, if, mypackage).
- Uppercase is where all the letters in a word are written in capitals. When there are more than two words in the name use underscores to separate them (e.g., MAX_HOURS)
- CamelCase (also known as Upper CamelCase) is where each new word begins with a capital letter (e.g., ActionListener)
- Mixed case (also known as Lower CamelCase) is the same as CamelCase except the first letter of the name is in lowercase (e.g., hasChildren, customerFirstName, customerLastName).

Java literals

- Java Literals are syntactic representations of boolean, character, numeric, or string data. Literals provide a means of expressing specific values in your program.

- For example, in the following statement, an integer variable named count is declared and
- assigned an integer value.

- Literal: Any constant value which can be assigned to the variable is called as literal/constant.

```
// Here 100 is a constant/literal.
int x = 100;
String s =
"Hello"; float
a = 101.230;
char ch = 'a';
```

Taking input from the console

There are 3 ways to get input from the console:-

1. Using Buffered Reader Class
2. Using Scanner Class
3. Using Console Class

1. Using Buffered Reader Class

This is the Java classical method to take input, Introduced in JDK1.0. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

Advantages

The input is buffered for efficient reading.

Drawback:

The wrapping code is hard to remember.

Check program: BufferReaderDemo.java

1. Using Buffered Reader Class

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class BufferedReaderDemo {
public static void main(String args[]) throws IOException{
InputStreamReader reader = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(reader);
System.out.println("What is your name?");
String name=br.readLine();
System.out.println("Welcome "+name);
}}
```

2. Using Scanner Class

This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however it is also can be used to read input from the user in the command line.

Advantages:

- Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
- Regular expressions can be used to find tokens.

Disadvantages :

- The reading methods are not synchronized

2. Using Scanner Class

```
import java.util.Scanner;
/*package whatever //do not write package name here */
class ScannerDemo {
public static void main (String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter your number");
int t = sc.nextInt();
System.out.println("Number you entered is: " + t);
}}
```

2. Using Scanner Class

Marwadi University

Method	Description
nextByte()	Accepts a byte
nextShort()	Accepts a short
nextInt()	Accepts an int
nextLong()	Accepts a long
next()	Accepts a single word
nextLine()	Accept a line of String
nextBoolean()	Accepts a boolean
nextFloat()	Accepts a float
nextDouble()	Accepts a double

Check program: ScannerClassDemo.java

3. Using Console Class

It has been becoming a preferred way for reading user's input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).

Advantages:

Reading password without echoing the entered characters.
Reading methods are synchronized.
Format string syntax can be used.

Drawback:
Does not work in non-interactive environment (such as in an IDE).

Check program: ConsoleClassDemo.java

3. Using Console Class

```
public class Main{
public static void main(String[] args)
{
// Using Console to input data from user
System.out.println("Enter your data");
String name = System.console().readLine();
System.out.println("You entered: "+name);
}}
```

Operators in Java

Operator in Java is a symbol which is used to perform operations.
For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

1. Arithmetic operators (+,-,*,/,%,++,--)
2. Assignment operators (=,+=,=,*=,/=,%=,&=,|=,^=,>>=,<<=)
3. Comparison operators (==,!=,>,<,>=,<=)
4. Short Circuit Logical operators (&&,||,!)
5. Bitwise operators (&,|,^,~,<<,>>,>>>)
6. Ternary Operator (?:)

Operators Precedence & Associativity

Operators	Precedence	Associativity
postfix increment and decrement	++, --	left to right
prefix increment and decrement, and unary	++, --, +, -, ~, !	right to left
multiplicative	*, /, %	left to right
additive	+, -	left to right
shift	<<, >>, >>>	left to right
relational	<, >, <=, >=	left to right
equality	==, !=	left to right

Operators Precedence & Associativity

Operators	Precedence	Associativity
bitwise AND	&	left to right
bitwise exclusive OR	^	left to right
bitwise inclusive OR	|	left to right
logical AND	&&	left to right
logical OR	||	left to right
ternary	? :	right to left
assignment	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=	left to right

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform
various operations i.e.:

incrementing/decrementing a value by one
negating an expression
inverting the value of a boolean

```
postfix  expr++ expr--
prefix   ++expr --expr +expr -expr ~ !
```

Java Unary Operator

```
class OperatorExample{
public static void main(String args[]){
int x=10
System.out.println(x++);
System.out.println(++x);
System.out.println(x--);
System.out.println(--x);
}}
```

```
Output:

10
12
12
10
```

Java Unary Operator

```
class OperatorExample{
   public static void main(String args[]){
   int a=10;
   int b=10;
   System.out.println(a++ + ++a);
   System.out.println(b++ + b++);

   }}
```

```
Output:

22
21
```

Java Arithmetic Operators

Java arithmatic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

```
class OperatorExample{
public static void main(String args[])
int a=10;
int b=5;
System.out.println(a+b);
System.out.println(a-b);
System.out.println(a*b);
System.out.println(a/b);
System.out.println(a%b);
}}
```

Output:

```
15
5
50
2
0
```

Java Arithmetic Operators

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10*10/5+3-1*4/2);
}}
```

```
Output:

21
```

Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);
System.out.println(10<<3);
System.out.println(20<<2);
System.out.println(15<<4);
}}
```

Output:

```
40
80
80
240
```

Java Right Shift Operator

Marwadi University

The Java right shift operator >> is used to move left operands value to right by the number of bits specified by the right operand.

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);
System.out.println(20>>2);
System.out.println(20>>3);
}}
```

Output:

```
2
5
2
```

Java AND operator

Example: Logical && and Bitwise &

The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[]){
int a=10; int b=5;  int c=20;
System.out.println(a<b&&a<c);
System.out.println(a<b&a<c);
}}
```

Output:

```
false
false
```

Java OR operator

Example: Logical || and Bitwise |

The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[]){
 int a=10; int b=5; int c=20;
System.out.println(a>b||a<c);
System.out.println(a>b|a<c);
System.out.println(a>b||a++<c);
System.out.println(a);
System.out.println(a>b|a++<c);
System.out.println(a);
}}
```

Output:

```
true
true
true
10
true
11
```

Java Ternary operator

Java Ternary operator is used as one liner replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

```
class OperatorExample{
public static void main(String args[]){
int a=2;  int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
Output: 2
```

Java Assignment operators

Java assignment operator is one of the most common operator. It is used to assign the value on its right to the operand on its left.

```
class OperatorExample{
public static void main(String args[]){
int
a=10;
int
b=20;
a+=4;
b-=4;
System.out.println(a);
System.out.println(b);
}}
```

Output:

```
14
16
```

Casting in Java

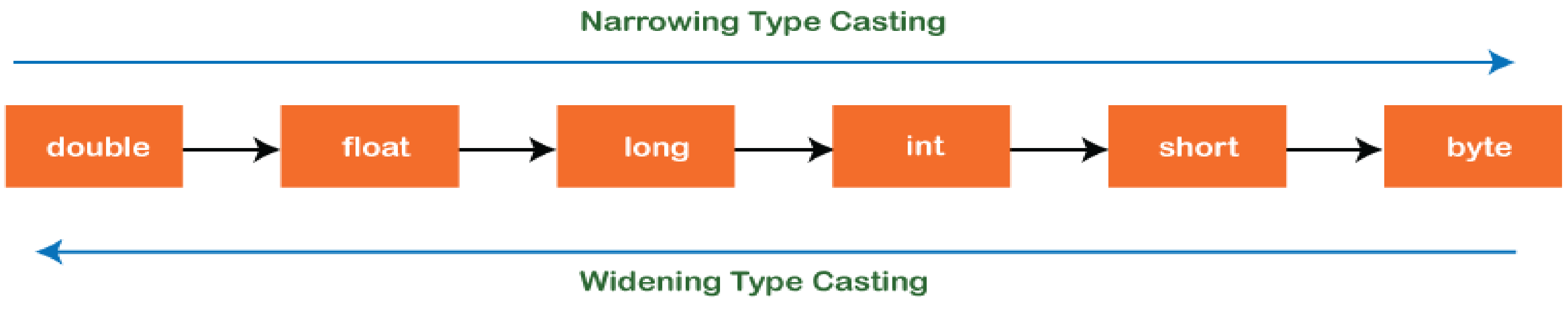
There are two types of type casting:

- **Widening Type Casting**

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is **no chance to lose data**.

- **Narrowing Type Casting**

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer.



Type Casting in Java

Casting between char and numeric types

- When a **floating-point** value is cast into a **char**, the floating-point value is first cast into an int, which is then cast into a char.
  - char ch = (char)65.25; // Decimal 65 is assigned to ch
  - System.out.println(ch); // ch is character A

- When a char is cast into a numeric type, the character's Unicode is cast into the specified numeric type.
  - int i = (int)'A'; // The Unicode of character A is assigned to i
  - System.out.println(i); // i is 65

- Implicit casting can be used if the result of a casting fits into the target variable. Otherwise, explicit casting must be used.
  - byte b = 'a';
  - int i = 'a';

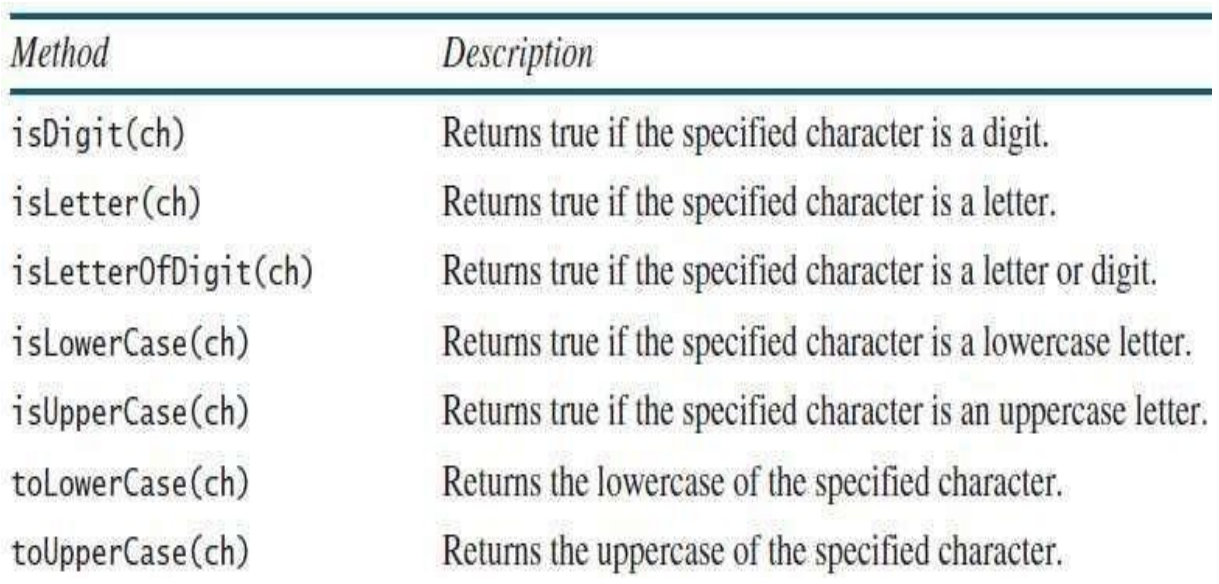
Casting between char and numeric types

- int i = '2' + '3'; // (int)'2' is 50 and (int)'3' is 51
- System.out.println("i is " + i); // i is 101
- int j = 2 + 'a'; // (int)'a' is 97
- System.out.println("j is " + j); // j is 99
- System.out.println(j + " is the Unicode for character " + (char)j); // 99 is the Unicode for character c
- System.out.println("Chapter " + '2');
- **OUTPUT** : i is 101

```
i is 101
j is 99
99 is the Unicode for characterc
Chapter 2
```

Casting between char and numeric types

- System.out.println("isDigit('a') is " + Character.isDigit('a')); //false
- System.out.println("isLetter('a') is " + Character.isLetter('a')); //true
- System.out.println("isLowerCase('a') is "+ Character.isLowerCase('a')); //true



Method	*Description*
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOfDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

Casting in Java

```
class Main {
 public static void main(String[] args) {
   // create int type variable
   int num = 10;
   System.out.println("The integer value: " + num);

   // convert into double type - Implicit
   double data = num;
   System.out.println("The double value: " + data);
 }
}
```

Casting in Java

```
class Main {
  public static void main(String[] args) {
    // create double type variable
    double num = 10.99;
    System.out.println("The double value: " + num);

    // convert into int type - Explicit
    int data = (int)num;
    System.out.println("The integer value: " + data);
  }
}
```

Summary

- History & Features of Java
- Java Virtual Machine
- Java Runtime Environment
- Bytecode
- Objected Oriented principles
- Datatypes
- Variables
- Final keyword
- Operators & precedence
- Scanner class for input
- Type conversion

END OF UNIT - 1