Core Java

Akshita Chanchlani

Batch Code : CJ-0-08

Start Date : 7th Feb 2022

End Date : 28th Feb 2022

Timings : 9:00am to 1:30pm
(Mon-Fri)

Course Contents

- Java Language Features, Wrapper Classes
- Class, Reference, Instance , Static initialization block
- Package, Enum, And Array
- String Handling
- Exception Handling
- Association and Inheritance, Abstract classes and Interface
- Collection Framework
- Functional Programming
- File Handling
- Reflection
- Multithreading

Trainer Introduction

- **Name**: Mrs.Akshita S.Chanchlani
- **Designation** : Associate Head SBU Technical
- **Education** :
  - PhD (Pursuing) : Computer Science and Engineering
  - Masters of Engineering (ME) in Information Technology
  - B.Tech in Computer Engineering, From VJTI Mumbai
- **Training Experience**
  - PreCAT and PG Course Training at Sunbeam
  - C, C++ , Operating System, Networks, Core Java, Python
- **Professional Experience**
  - **12+ years**
- **Email** : akshita.chanchlani@sunbeaminfo.com

Day01 Agenda

- Documentation
- Java history
- Version history
- Java platforms , SDK, JDK, JRE, JVM
- Components of Java platform
- C++ versus Java terminology
- Core Java terminologies
- Structure of Java class
- Java application flow of execution
- Comments and Entry point method
- Data types Wrapper class Introduction
- Narrowing and Widening
- Boxing and Unboxing
- Command Line Argument
- Operators
- Control Statements
- Scanner Class

Java Text Books

- Java, The complete reference, Herbert Schildt
- Core and Advanced Java Black Book
- Head First Java: A Brain-Friendly Guide , by Kathy Sierra

Documentation

- **JDK download:**
  - https://adoptopenjdk.net/
- **Spring Tool Suite 3 download:**
  - https://github.com/spring-projects/toolsuite-distribution/wiki/Spring-Tool-Suite-3
- **Oracle Tutorial:**
  - https://docs.oracle.com/javase/tutorial/
- **Java 8 API and Java 11 API Documentation Documentation:**
  - https://docs.oracle.com/javase/8/docs/api/
  - https://docs.oracle.com/en/java/javase/11/docs/api/
- **MySQL Connector:**
  - https://downloads.mysql.com/archives/c-j/
- **Core Java Tutorials:**
  1. http://tutorials.jenkov.com/java/index.html
  2. https://www.baeldung.com/java-tutorial
  3. https://www.journaldev.com/7153/core-java-tutorial

Java Installation

1. JDK 11 download
https://www.oracle.com/java/technologies/javase-jdk11-downloads.html
Choose your platform , download the installer and install the JDK

2. Open command prompt or terminal
Type

```
java -version
```

It should show you : java version "11.0.7" 2020-04-14 LTS
Note : Basic version should be 11, update version may differ.

3.Java IDE download
https://github.com/spring-projects/toolsuite-distribution/wiki/Spring-Tool-Suite-3
Choose 3.9.16 version and download. Extract it. No installation is required.

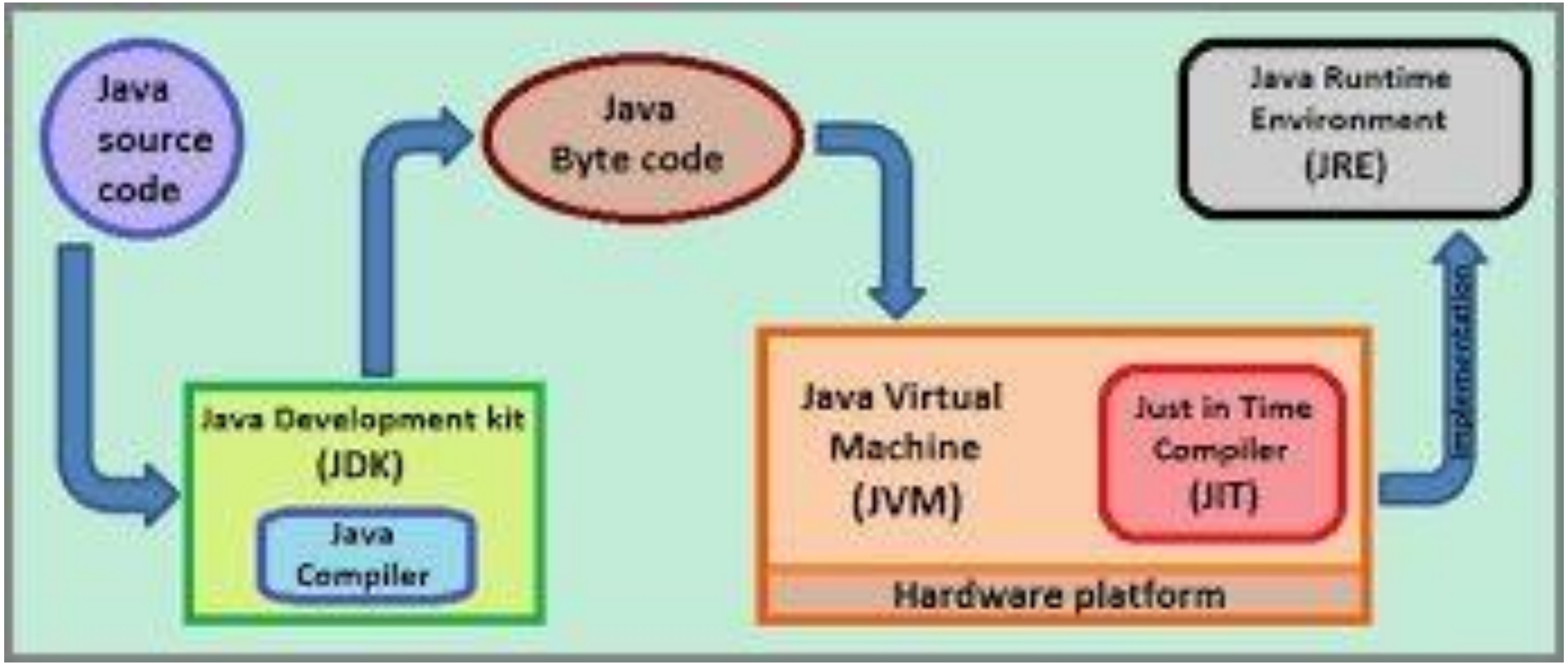
4. Java API Dcoumentation link
https://docs.oracle.com/en/java/javase/11/docs/api/index.html

5. Offline version
https://www.oracle.com/java/technologies/javase-jdk11-doc-downloads.html

WHY Java ?

1. Platform or architecture independent (Write once run anywhere!)
2. Simple & robust
3. Secure
4. Automatic memory management.
5. Inherent Multi threaded support
6. Object Oriented support -- Encapsulation, Inheritance & polymorphism , abstraction.
7. Excellent I/O support
8. Inherent networking support for TCP/IP , UDP/IP programming & for URLs
9. Adds functional programming support.

Development & Execution of java application.


Java source code
Java Development kit (JDK)
Java Compiler
Java Byte code
Java Virtual Machine (JVM)
Just in Time Compiler (JIT)
Hardware platform
implementation
Java Runtime Environment (JRE)

Java History

- **James Gosling, Mike Sheridan** and **Patrick Naughton** initiated the Java language project in June 1991.

- **Java** was originally **developed by James Gosling** at **Sun Microsystems** and **released in 1995.**

- The language was initially called **Oak** after an oak tree that stood outside Gosling's office.

- Later the project went by the name ***Green*** and was finally renamed ***Java***, from Java coffee, a type of coffee from Indonesia.

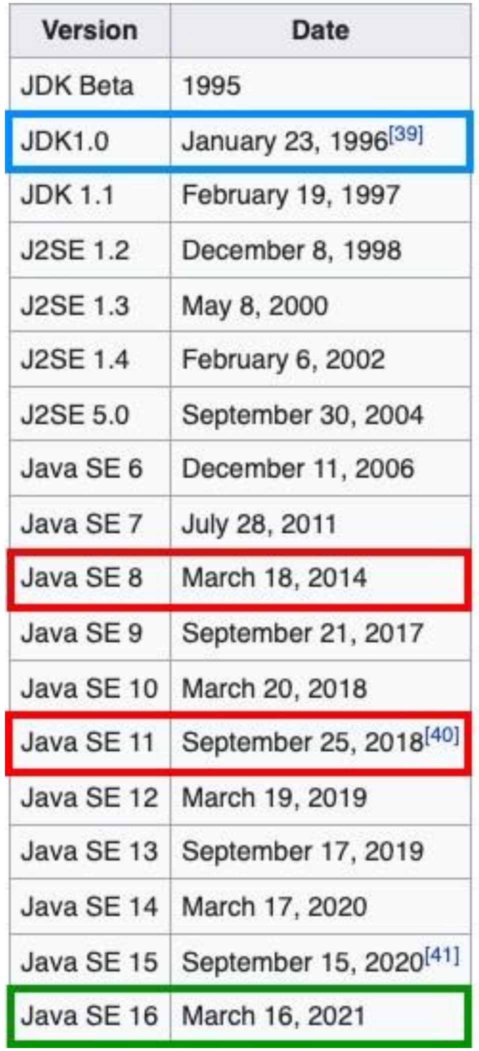
- Gosling and his team did a brainstorm session and after the session, they came up with several names such as **JAVA, DNA, SILK, RUBY, etc.**

- Sun Microsystems released the first public implementation as Java 1.0 in 1996.

Java History

- The Java programming language is a general-purpose, concurrent, class-based, object-oriented language.
- The Java programming language is a high-level language.
- The Java programming language is related to C and C++ but is organized rather differently, with a number of aspects of C and C++ omitted and a few ideas from other languages included.
- **It is intended to be a production language, not a research language.**
- The Java programming language is statically typed.
- It promised **Write Once, Run Anywhere** (**WORA**) functionality.

Version History



Version	Date
JDK Beta	1995
JDK1.0	January 23, 1996[39]
JDK 1.1	February 19, 1997
J2SE 1.2	December 8, 1998
J2SE 1.3	May 8, 2000
J2SE 1.4	February 6, 2002
J2SE 5.0	September 30, 2004
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8	March 18, 2014
Java SE 9	September 21, 2017
Java SE 10	March 20, 2018
Java SE 11	September 25, 2018[40]
Java SE 12	March 19, 2019
Java SE 13	September 17, 2019
Java SE 14	March 17, 2020
Java SE 15	September 15, 2020[41]
Java SE 16	March 16, 2021

- The first version was released on January 23, 1996.

- The acquisition of Sun Microsystems by Oracle Corporation was completed on January 27, 2010

- As of September 2020, Java 8 and 11 are supported as Long Term Support (LTS) versions

- In September 2017, Mark Reinhold, chief Architect of the Java Platform, proposed to change the release train to "one feature release every six months".

- OpenJDK (Open Java Development Kit) is a free and open source implementation of the (Java SE). It is the result of an effort Sun Microsystems began in 2006.

- Java Language and Virtual Machine Specifications: https://docs.oracle.com/javase/specs/

Java Platforms

1. **Java SE**
   - Java Platform Standard Edition.
   - It is also called as Core Java.
   - For general purpose use on Desktop PC's, servers and similar devices.
2. **Java EE**
   - Java Platform Enterprise Edition.
   - It is also called as advanced Java / enterprise java / web java.
   - Java SE plus various API's which are useful client-server applications.
3. **Java ME**
   - Java Platform Micro Edition.
   - Specifies several different sets of libraries for devices with limited storage, display, and power capacities.
   - It is often used to develop applications for mobile devices, PDAs, TV set-top boxes and printers.
4. **Java Card**
   - A technology that allows small Java-based applications (applets) to be run securely on smart cards and similar small-memory devices.

SDK, JDK, JRE, JVM

- **SDK** = Development Tools + Documentation + Libraries + Runtime Environment.

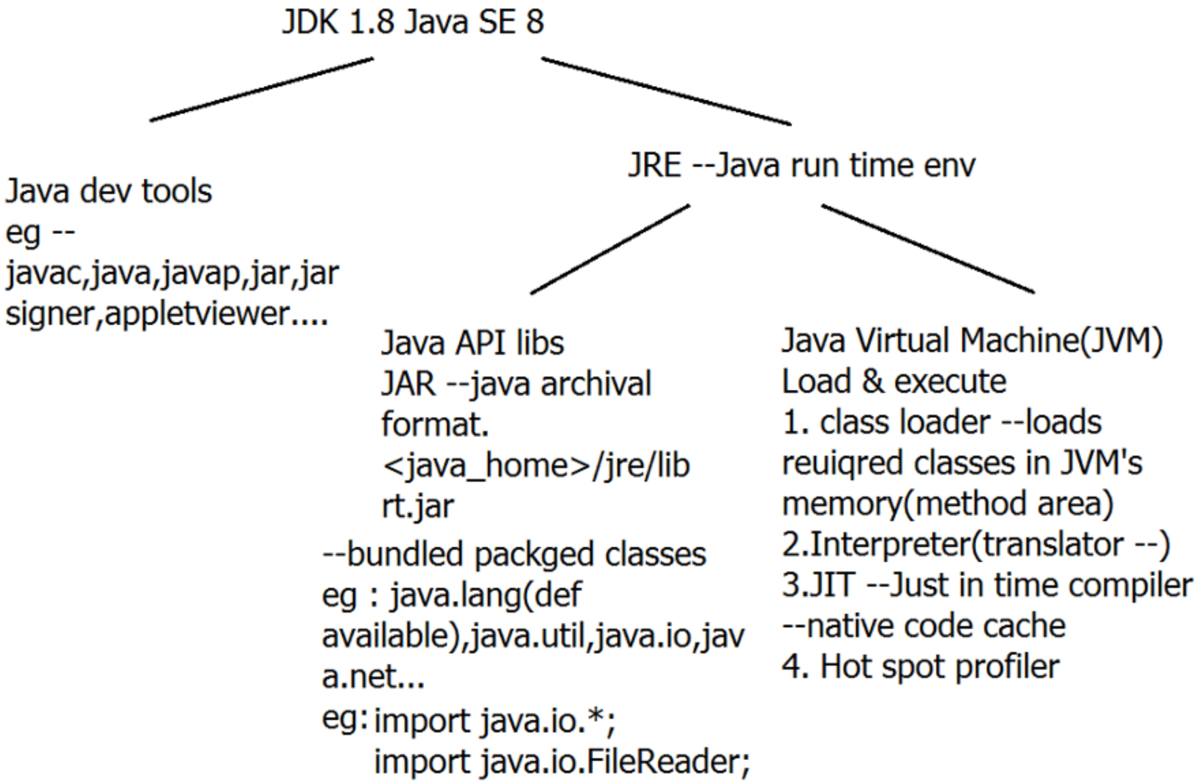
- **JDK** = Java Development Tools + Java Docs + **rt.jar** + JVM.
    - JDK : Java Development Kit.
    - It is a software, that need to be install on developers machine.
    - We can download it from oracle official website.

- **JDK** = Java Development Tools + Java Docs + **JRE**[**rt.jar + JVM**].
    - JRE : Java Runtime Environment.
    - rt.jar and JVM are integrated part of JRE.
    - JRE is a software which comes with JDK. We can also download it separately.
    - To deploy application, we should install it on client's machine.

- rt.jar file contains core Java API in **compiled form**.

- **JVM** : An engine, which manages execution of Java application. (also called as Execution Engine)

JDK



JDK 1.8 Java SE 8

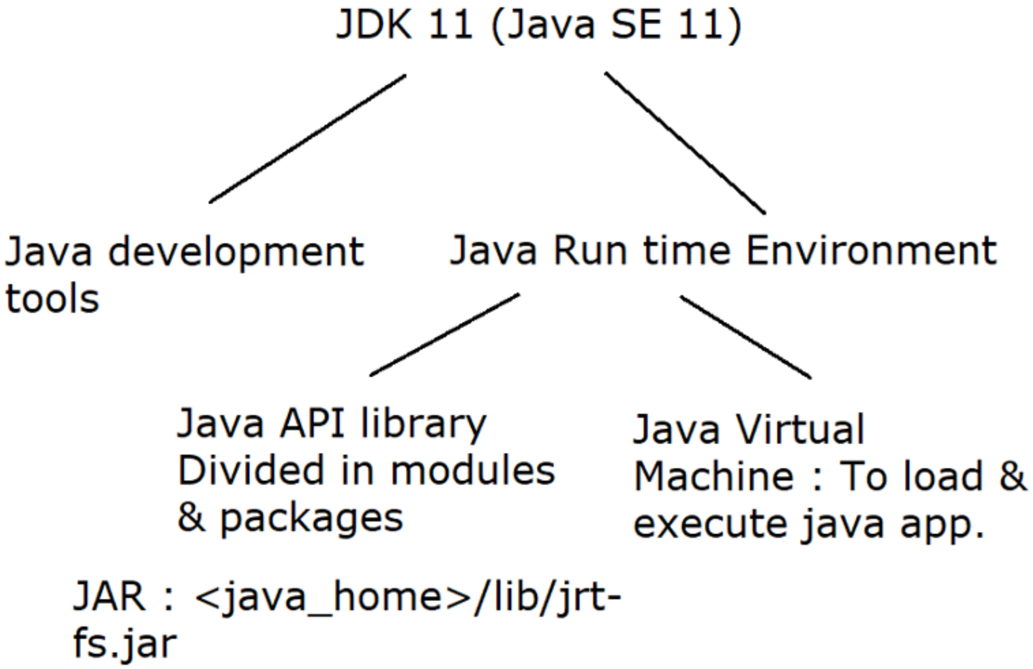
Java dev tools
eg --
javac,java,javap,jar,jar
signer,appletviewer....

JRE --Java run time env

Java API libs
JAR --java archival
format.
<java_home>/jre/lib
rt.jar
--bundled packged classes
eg : java.lang(def
available),java.util,java.io,jav
a.net...

```
eg: import java.io.*;
    import java.io.FileReader;
```

Java Virtual Machine(JVM)
Load & execute
1. class loader --loads
reuiqred classes in JVM's
memory(method area)
2.Interpreter(translator --)
3.JIT --Just in time compiler
--native code cache
4. Hot spot profiler



JDK 11 (Java SE 11)

Java development
tools

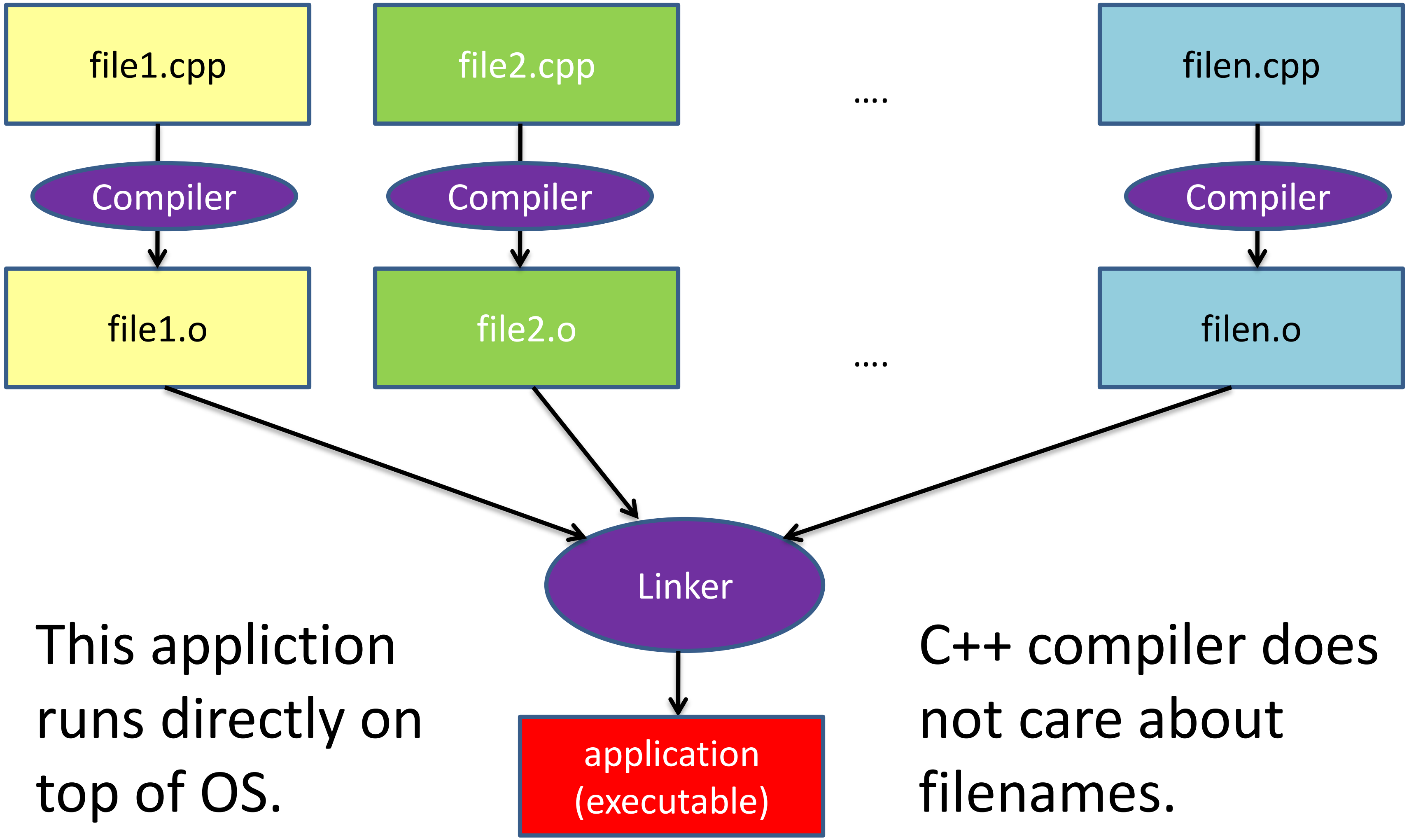
Java Run time Environment

Java API library
Divided in modules
& packages

Java Virtual
Machine : To load &
execute java app.

JAR : <java_home>/lib/jrt-
fs.jar

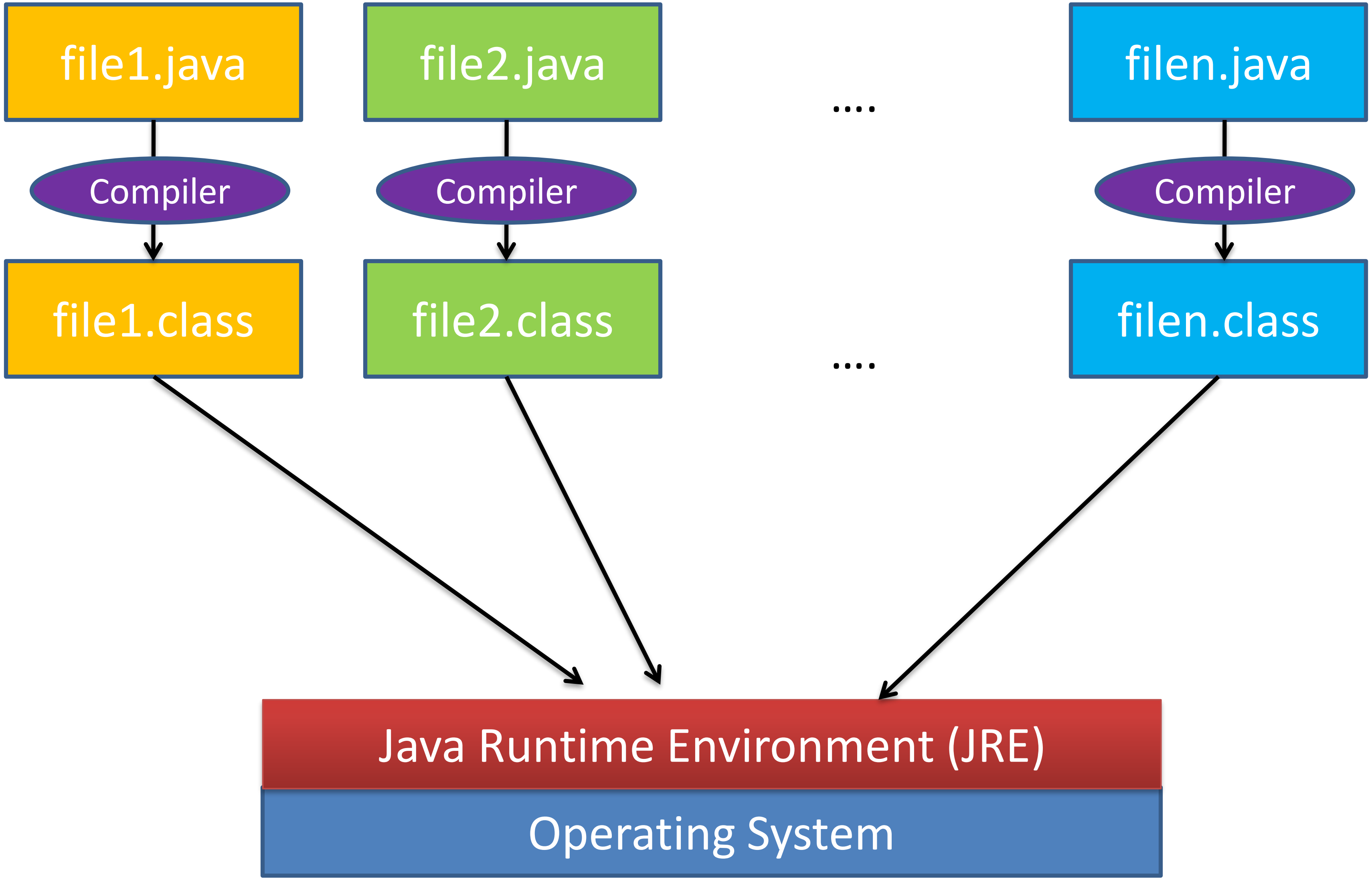
C++ compiler & Linker usage



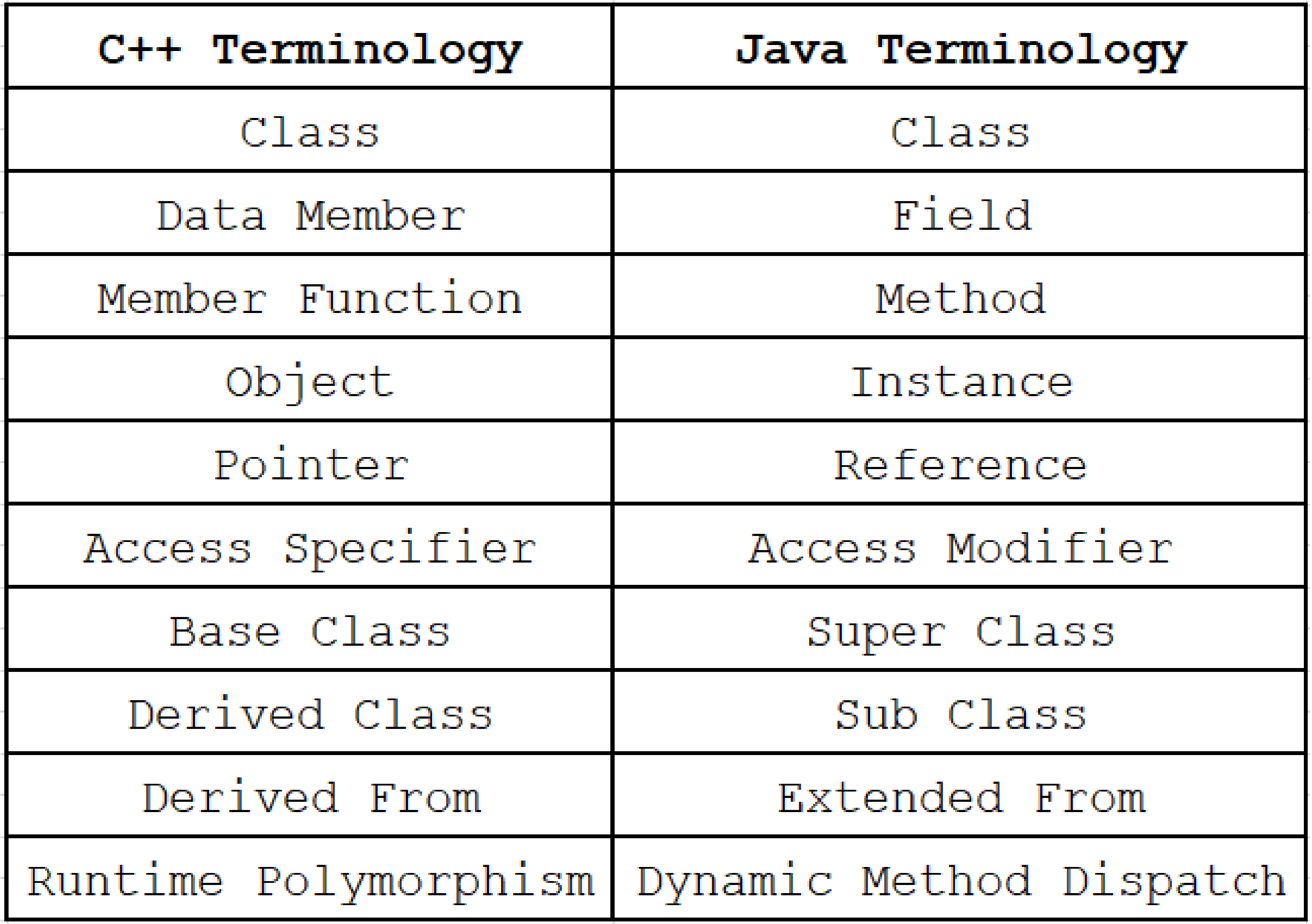
This appliction runs directly on top of OS.

C++ compiler does not care about filenames.

Java compiler usage



C++ versus Java terminology



C++ Terminology	Java Terminology
Class	Class
Data Member	Field
Member Function	Method
Object	Instance
Pointer	Reference
Access Specifier	Access Modifier
Base Class	Super Class
Derived Class	Sub Class
Derived From	Extended From
Runtime Polymorphism	Dynamic Method Dispatch

C++ VS Java

Development wise differences

- Java is platform independent language but c++ is dependent upon operating system. At compilation time Java Source code(.java) converts into byte code(.class) .The interpreter translates this byte code at run time into native code and gives output.
- Java uses both a compiler and interpreter, while C++ only uses a compiler

Syntactical differences

- 1. There is no final semi-colon at the end of the class definition.
- 2. Functions are called as methods.
- 3. main method is a member of class & has a fixed form
  - public static void main(String[] args) -- argument is an array of String. This array contains the command-line arguments.
- 4. main method must be inside some class (there can be more than one main function -- there can even be one in every class)
- 5. Like the C++ << operator,
  - To write to standard output, you can use either of the following:
    - System.out.println(...)
      - System.out.print(...) The + operator can be useful when printing.

C++ VS Java cont..

Features wise differences

- 1. C++ supports pointers whereas Java does not support pointer arithmetic.
- 2. C++ supports operator overloading , multiple inheritance but java does not.
- 3. C++ is nearer to hardware than Java.
- 4. Everything (except fundamental or primitive types) is an object in Java (Single root hierarchy as everything gets derived from java.lang.Object).
- 5. Java is similar to C++ but it doesn't have the complicated aspects of C++, such as pointers, templates, unions, operator overloading, structures, etc. Java also does not support conditional compilation (#ifdef/#ifndef type).
- 6. Thread support is built into Java but not in C++. C++11, the most recent iteration of the C++ programming language, does have Thread support though.
- 7. Internet support is built into Java, but not in C++. On the other hand, C++ has support for socket programming which can be used.
- 8.Java does not support header files and library files. Java uses import to include different classes and methods.
- 9.Java does not support default arguments.
- 10. There is no scope resolution operator :: in Java. It has . (dot operator) using which we can qualify classes with the namespace they came from.
- 11. There is no goto statement in Java.
- 12. Because of the lack of destructors in Java, exception and auto garbage collector handling is different than C++.
- 13. Java has method overloading, but no operator overloading unlike C++.
- 14. The String class does use the + and += operators to concatenate strings and String expressions use automatic type conversion,
- 15. Java is pass-by-value.
- 16. Java does not support unsigned integers.

Language Basics

- Keywords
- Variables
- Conditional Statements
- Loops
- Data Types
- Operators
- Coding Conventions

Java Keywords

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

Simple Hello Application

```
//File Name : Program.java
class Program{
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

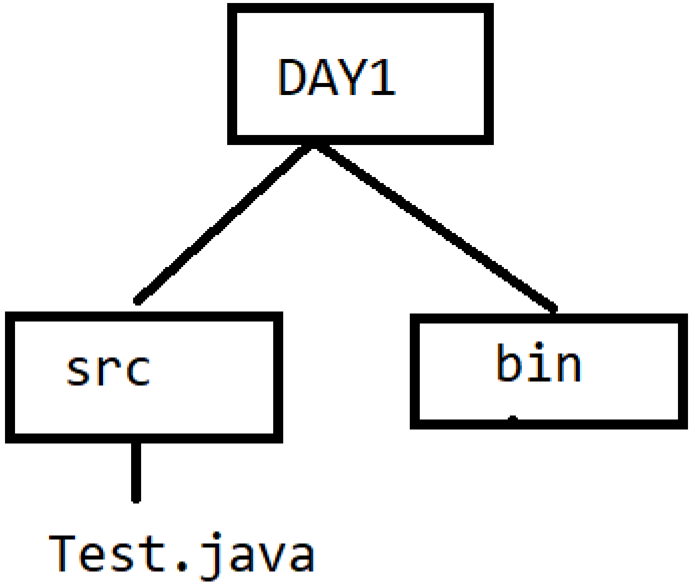
```
Compilation=> javac Program.java  //Output : Program.class
Execution=>   java Program
```

```
System    :   Final class declared in java.lang package
out       :   public static final field of System class. Type of out is PrintStream
println   :   Non static method of java.io.PrintStream class
```

To view class File :

javap -c Program.class

How to compile the code from src and store .class inside bin



DAY1

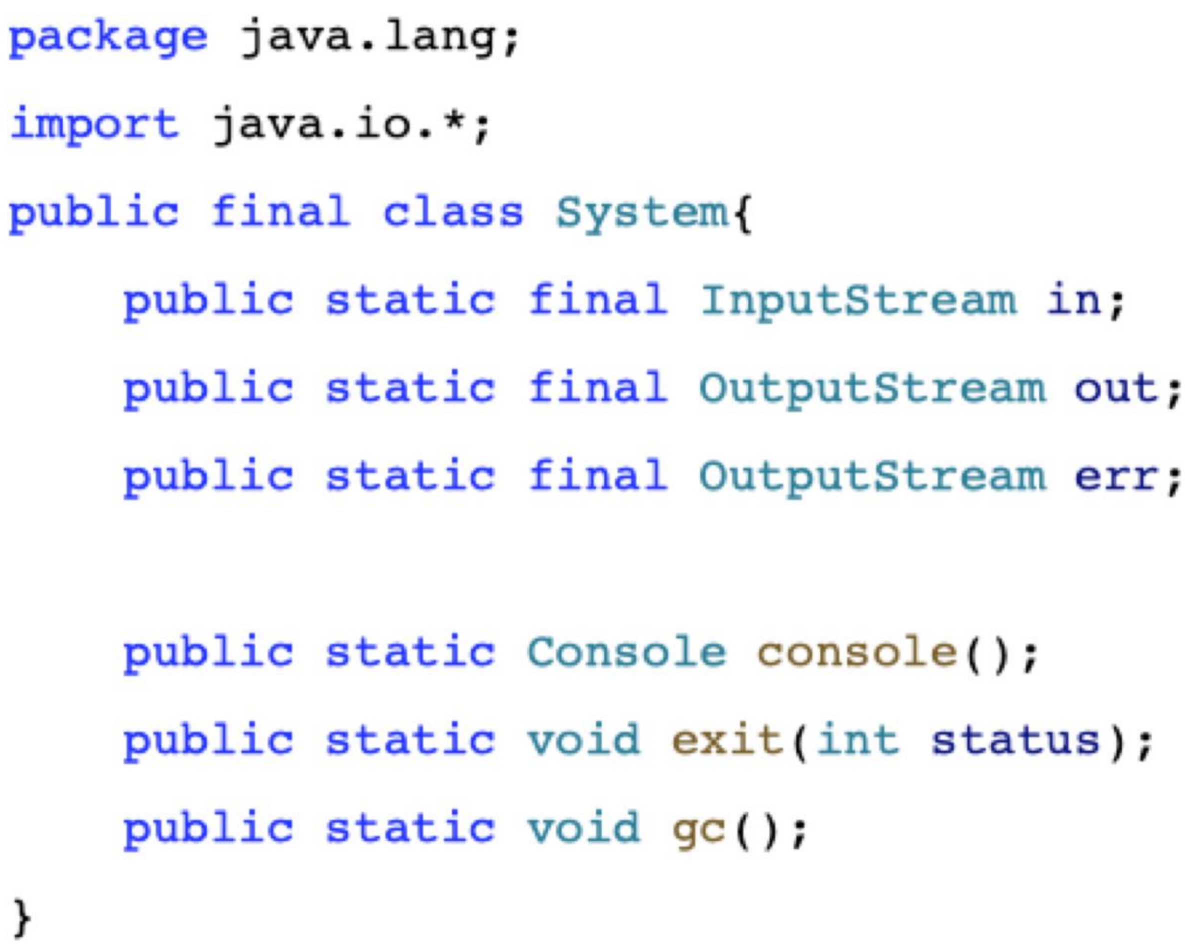
src

bin

Test.java

```
D:\DAY1> cd src
D:\DAY1\src>javac –d ..\bin Test.java
D:\DAY1\src>cd ..\bin
D:\DAY1\bin>java Test
```

java.lang.System class



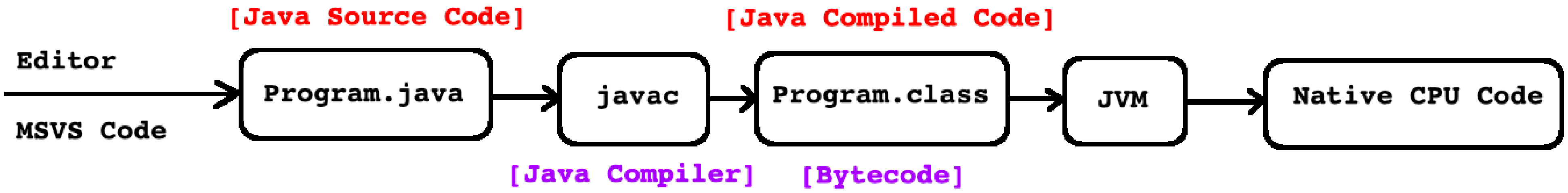
```java
package java.lang;
import java.io.*;
public final class System{
    public static final InputStream in;
    public static final OutputStream out;
    public static final OutputStream err;

    public static Console console();
    public static void exit(int status);
    public static void gc();
}
```

Stream

- Stream is an abstraction(object) which either produce(write)/consume(read) information from source to destination.
- Standard stream objects of Java which is associated with console:

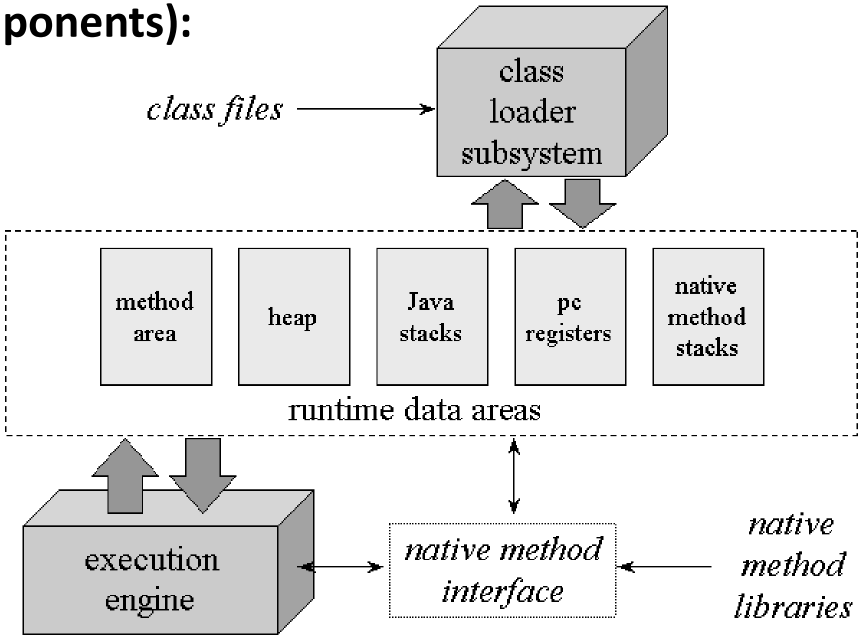
  1. **System.in**
     - Ø It represents keyboard.
  2. **System.out**
     - Ø It represents Monitor.
  3. **System.err**
     - Ø Error stream which represents Monitor.

Java application flow of execution



Components of JVM (Major 3 Components):

1. **Class Loader Sub System**
   - Bootstrap class loader
   - Extension classLoader
   - Application classLoader
   - User Defined class Loader
2. **Runtime Data Areas**
3. **Execution Engine**
   - Interpreter
   - Compiler
   - Garbage Collector



[JVM Architecture]

Method Area

- Metaspace: JDK 1.8 onwards Loaded Byte codes (Loaded class info) 1 Single copy static data members, constructors, methods

Heap

- Java object heap state of the object (non static data members) 1 single copy

Java Stack

- Stack is created one per thread , method local info(like args,local vars, ret vals) Individual stack : stack frames

Comments

- Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain only information that is relevant to reading and understanding the program.
- Types of Comments:
  - **Implementation Comment**
    - Implementation comments are those found in C++, which are delimited by /*...*/, and //.

      1. Single-Line Comment
      2. Block Comment(also called as multiline comment)

  - **Documentation Comment**
    - Documentation comments (known as "doc comments") are Java-only, and are delimited by **/**...*/**.
    - Doc comments can be extracted to HTML files using the javadoc tool.

Entry point method

- Syntax:
  1. **public static void main(String[] args)**
  2. **public static void main(String... args)**

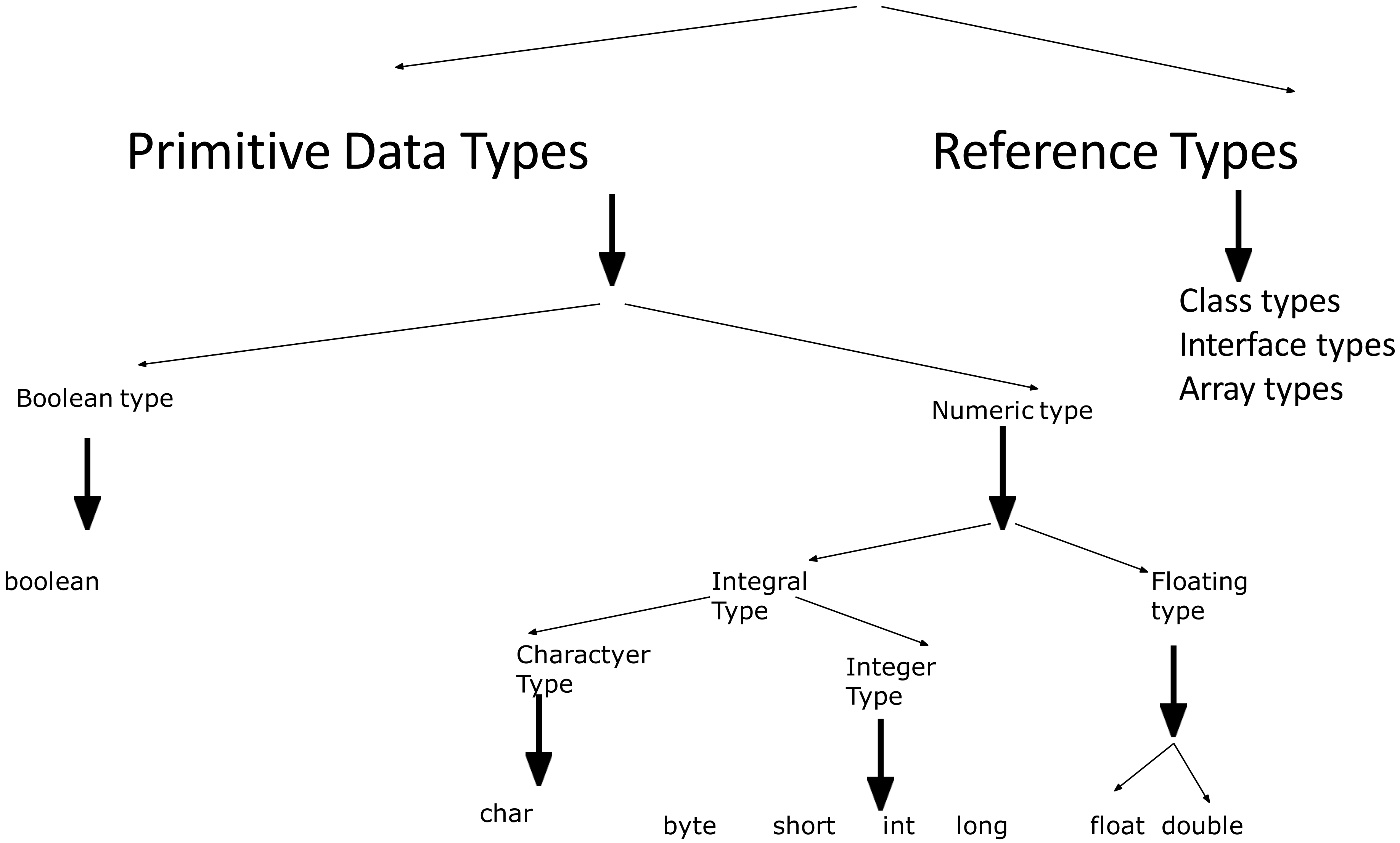
- Java compiler do not check/invoke main method. JVM invoke main method.

- When we start execution of Java application then JVM starts execution of two threads:
  1. Main thread : responsible for invoking main method.
  2. Garbage Collector : responsible for deallocating memory of unused object.

- We can overload main method in Java.

- We can define main method per class. But only one main method can be considered as entry point method.

Data types In Java



Primitive Data Types

Reference Types

Class types

Interface types

Array types

Boolean type

Numeric type

boolean

Integral Type

Floating type

Charactyer Type

Integer Type

char

byte short int long

float double

Data Types

- Integral Type
  - byte 8 bits
  - short 16 bits
  - int 32 bits
  - long 64 bits
- Textual Type
  - char 16 bits, UNICODE Character
  - String
- Floating Point Type
  - float 32 bits
  - double 64 bits
- Boolean Type 1 bit
  - true
  - false

Data Types

- Data type of any variable decide following things:
    1. **Memory:** How much memory is required to store the data.
    2. **Nature:** Which kind of data is allowed to store inside memory.
    3. **Operation:** Which operations are allowed to perform on the data stored in memory.
    4. **Range:** Set of values that we can store inside memory.

- The Java programming language is a statically typed language, which means that every variable and every expression has a type that is known at compile time.
    - Types of data type:
        1. **Primitive type(also called as value type)**
            - **boolean** type
            - Numeric type
                1. Integral types(**byte, char, short, int, long**)
                2. Floating point types(**float, double**
        2. **Non primitive type(also called as reference type)**
            - **Interface, Class, Type variable, Array**

Variables

- A **variable** is a name given memory location. That memory is associated to a data type and can be assigned a value.
- int n; float f1; char ch; double d;

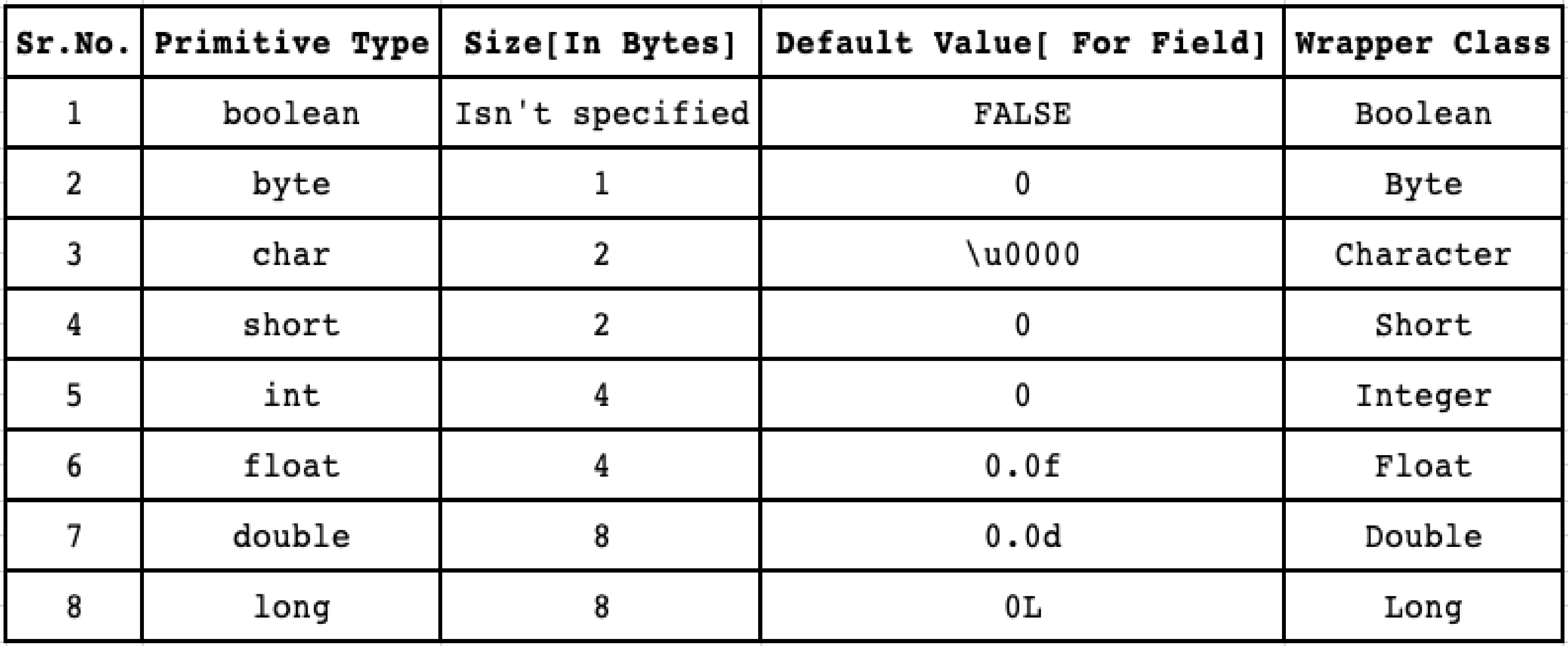
Rules of Variables

- All variable names must begin with a letter of the alphabet, an underscore (_), or a dollar sign ($). Can't begin with a digit. The rest of the characters may be any of those previously mentioned plus the digits 0-9.

- The convention is to always use a (lower case) letter of the alphabet. The dollar sign and the underscore are discouraged.

```
 1. int n1;
2. n1 =21 ;        // assignment
3.  int val=50; //initialization
5. double d = 21.8;   // initialization
6. d = n1;         // assignment
7. float f1 = 16.13F;
```

Data Types



Sr.No.	Primitive Type	Size[In Bytes]	Default Value[For Field]	Wrapper Class
1	boolean	Isn't specified	FALSE	Boolean
2	byte	1	0	Byte
3	char	2	\u0000	Character
4	short	2	0	Short
5	int	4	0	Integer
6	float	4	0.0f	Float
7	double	8	0.0d	Double
8	long	8	0L	Long

Note : Char datatype supports UNICODE character set, so 2 bytes .

Wrapper class

- In Java, primitive types are not classes. But for every primitive type, Java has defined a class. It is called wrapper class.

- All wrapper classes are final.

- All wrapper classes are declared in **java.lang** package.

- Uses of Wrapper class

   1. To parse string(i.e. to convert state of string into numeric type).

      example :

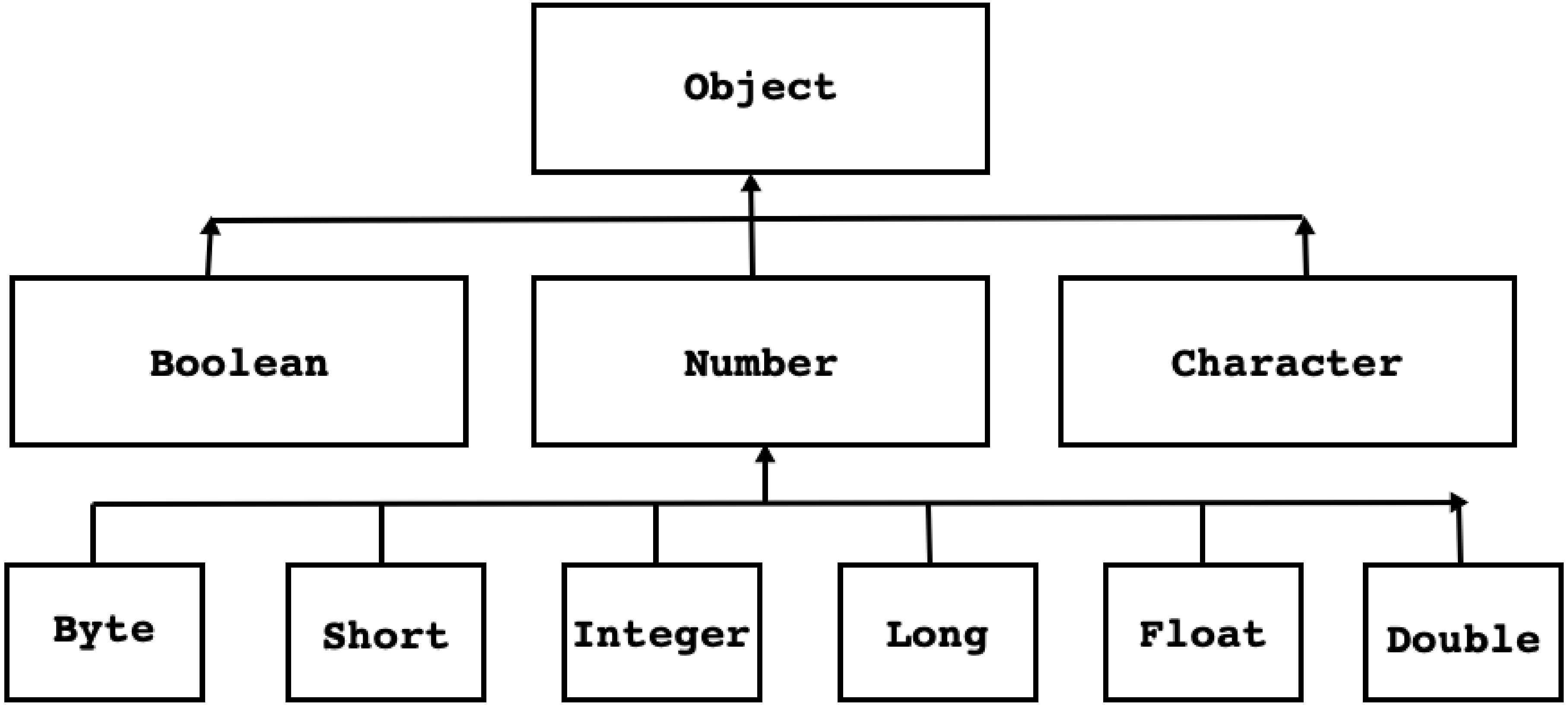
      ```
      int num = Integer.parseInt(“123”)
      float val = Float.parseFloat(“125.34f”);
      double d = Double.parseDouble(“42.3d”);
      ```

   1. To store value of primitive type into instance of generic class, type argument must be wrapper class.

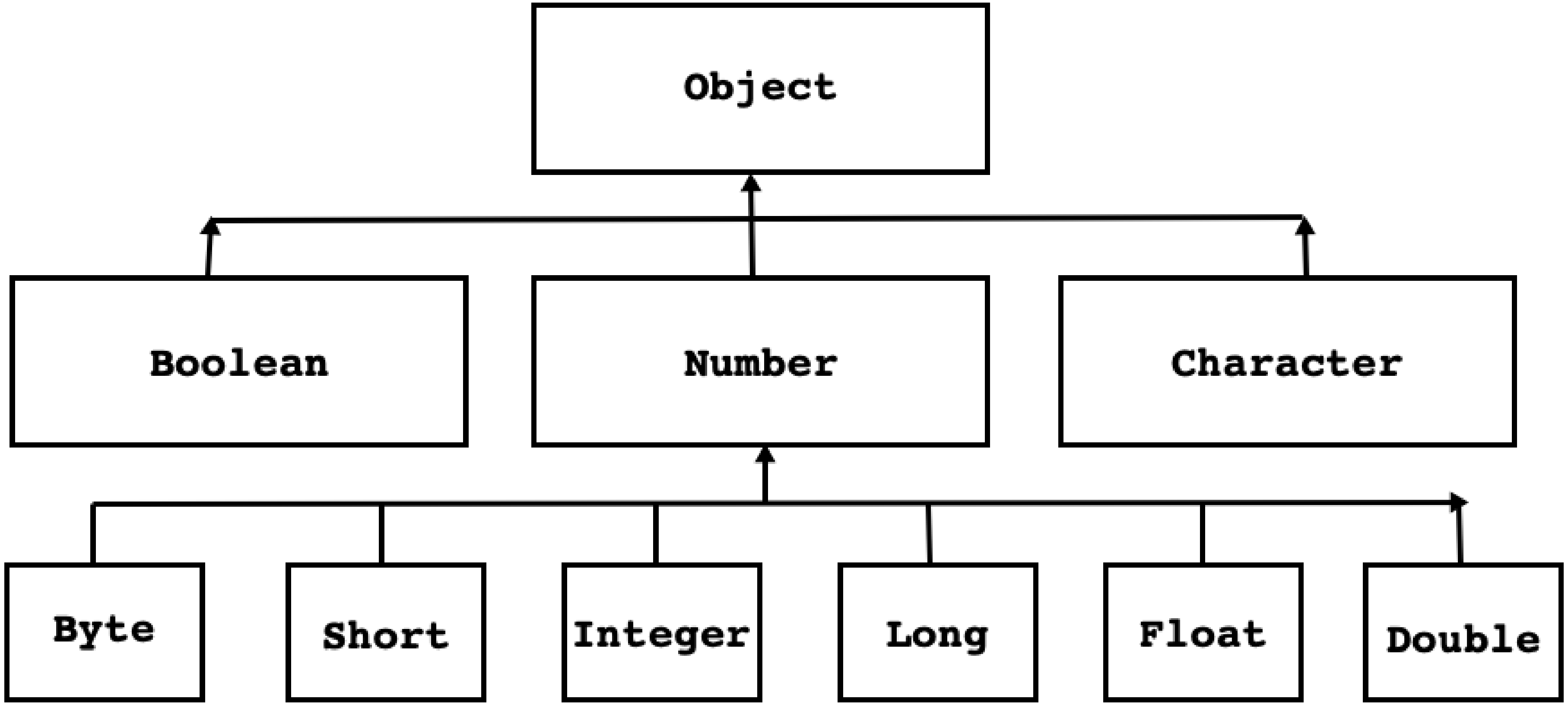
      ➢ **Stack<int> stk = new Stack<int>( );   //Not OK**

      ➢ **Stack<Integer> stk = new Stack<Integer>( );   //OK**

Wrapper class

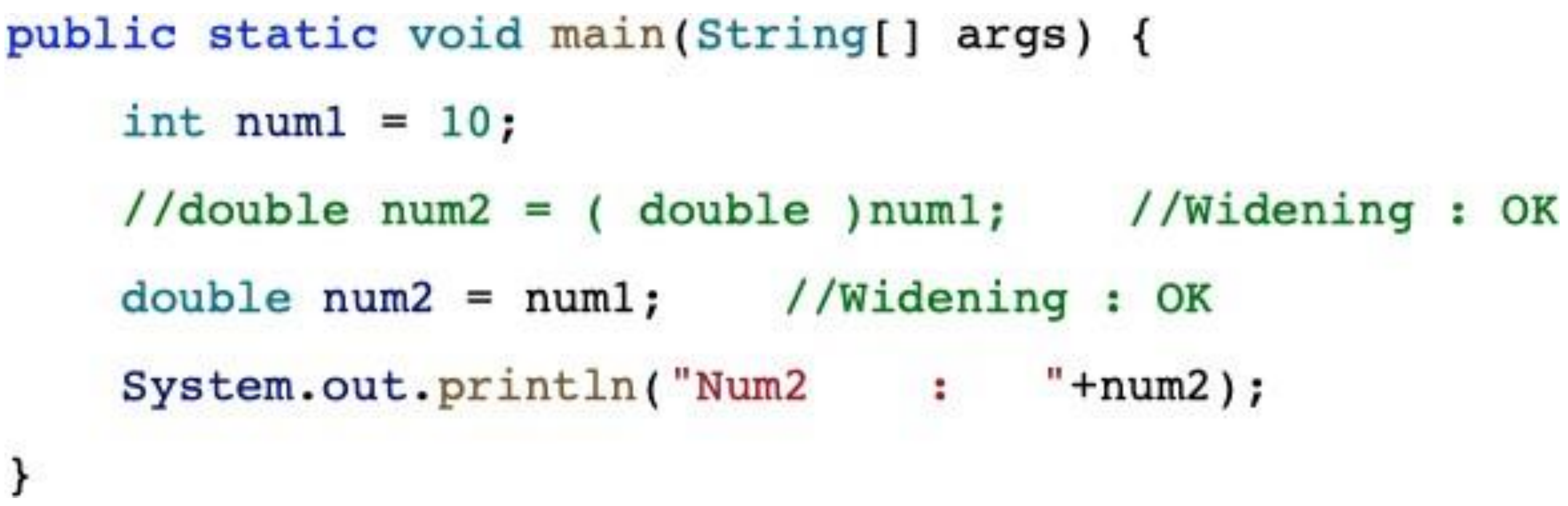


Wrapper class



Widening

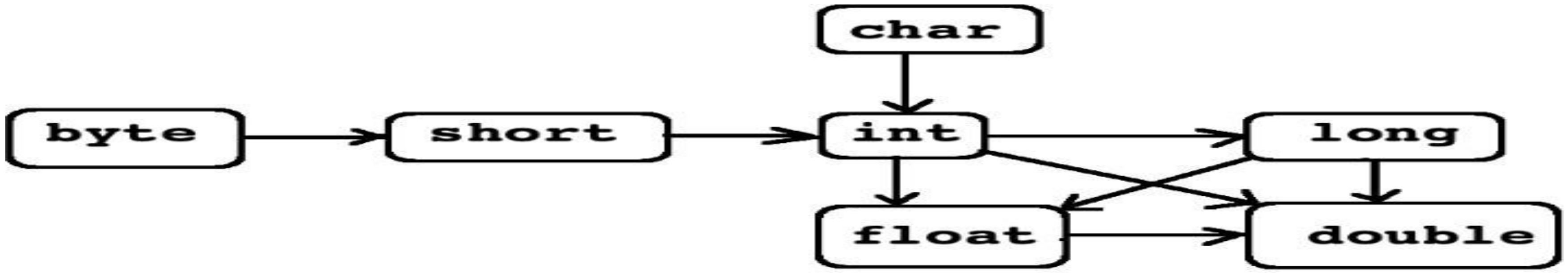
- Process of converting value of variable of narrower type into wider type is called widening.
- E.g. Converting int to double
-



```
public static void main(String[] args) {
    int num1 = 10;
    //double num2 = ( double )num1;    //Widening : OK
    double num2 = num1;    //Widening : OK
    System.out.println("Num2    :    "+num2);
}
```

- In case of widening, there is no loss of data
- So , explicit type casting is optional.

Widening



Widening Conversion

The range of values that can be represented by a float or double is much larger than the range that can be represented by a long. Although one might lose significant digits when converting from a long to a float, it is still a "widening" operation because the range is wider.

A widening conversion of an int or a long value to float, or of a long value to double, may result in loss of precision - that is, the result may lose some of the least significant bits of the value. In this case, the resulting floating-point value will be a correctly rounded version of the integer value, using IEEE 754 round-to-nearest mode.
Note that a double can exactly represent every possible int value.

long --->float ---is considered automatic type of conversion(since float data type can hold larger range of values than long data type)

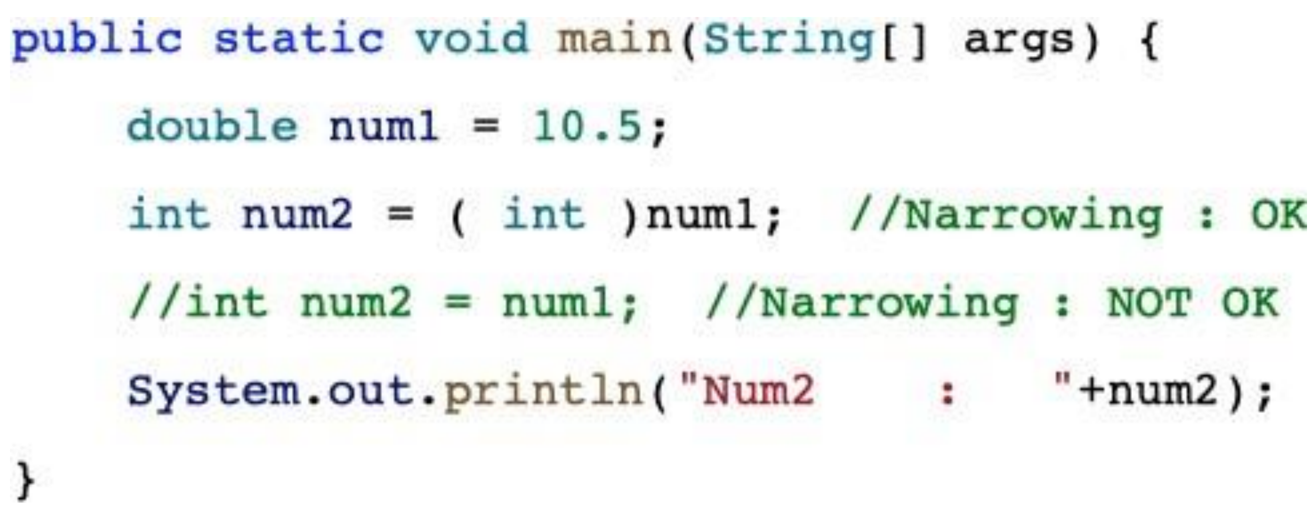
Rules

- src & dest - must be compatible, typically dest data type must be able to store larger magnitude of values than that of src data type.

- 1. Any arithmetic operation involving byte,short --- automatically promoted to --int
- 2. int & long ---> long
- 3. long & float ---> float
- 4. byte,short......& float & double----> double

Narrowing (Forced Conversion)

- Process of converting value of variable of wider type into narrower type is called narrowing.

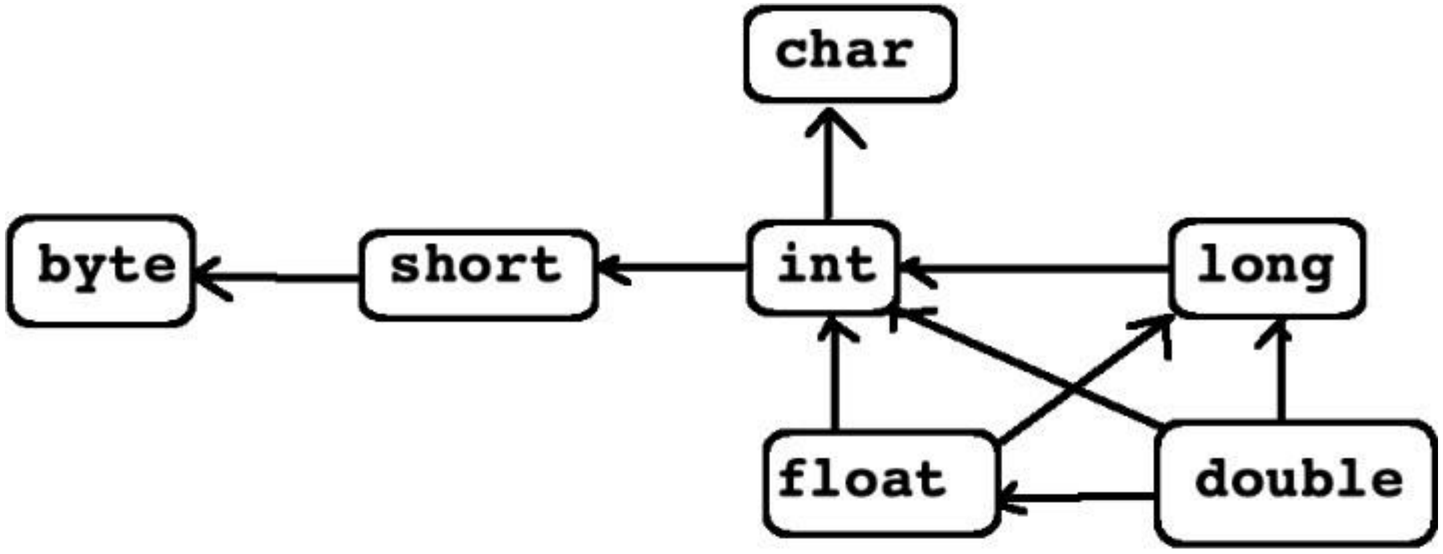


```
public static void main(String[] args) {
    double num1 = 10.5;
    int num2 = ( int )num1;  //Narrowing : OK
    //int num2 = num1;  //Narrowing : NOT OK
    System.out.println("Num2     :    "+num2);
}
```

- In case of narrowing, explicit type casting is mandatory.

Note : In case of narrowing and widening both variables are of primitive

Narrowing



Narrowing Conversion.

eg ---

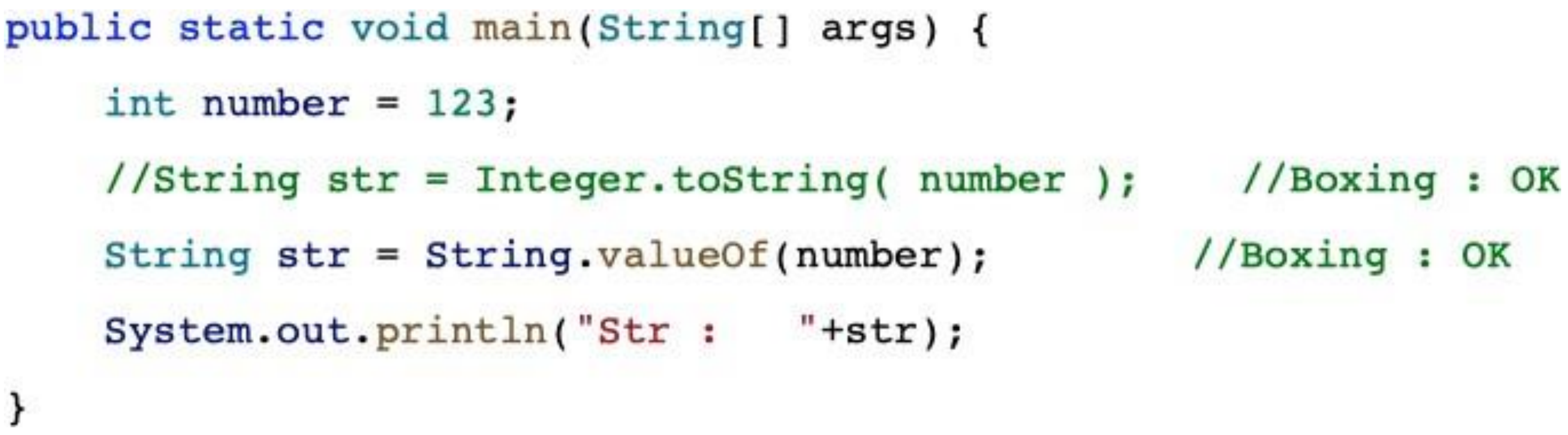
double ---> int

float --> long

double ---> float

Boxing

- Process of converting value of variable of primitive type into non primitive type is called **boxing.**

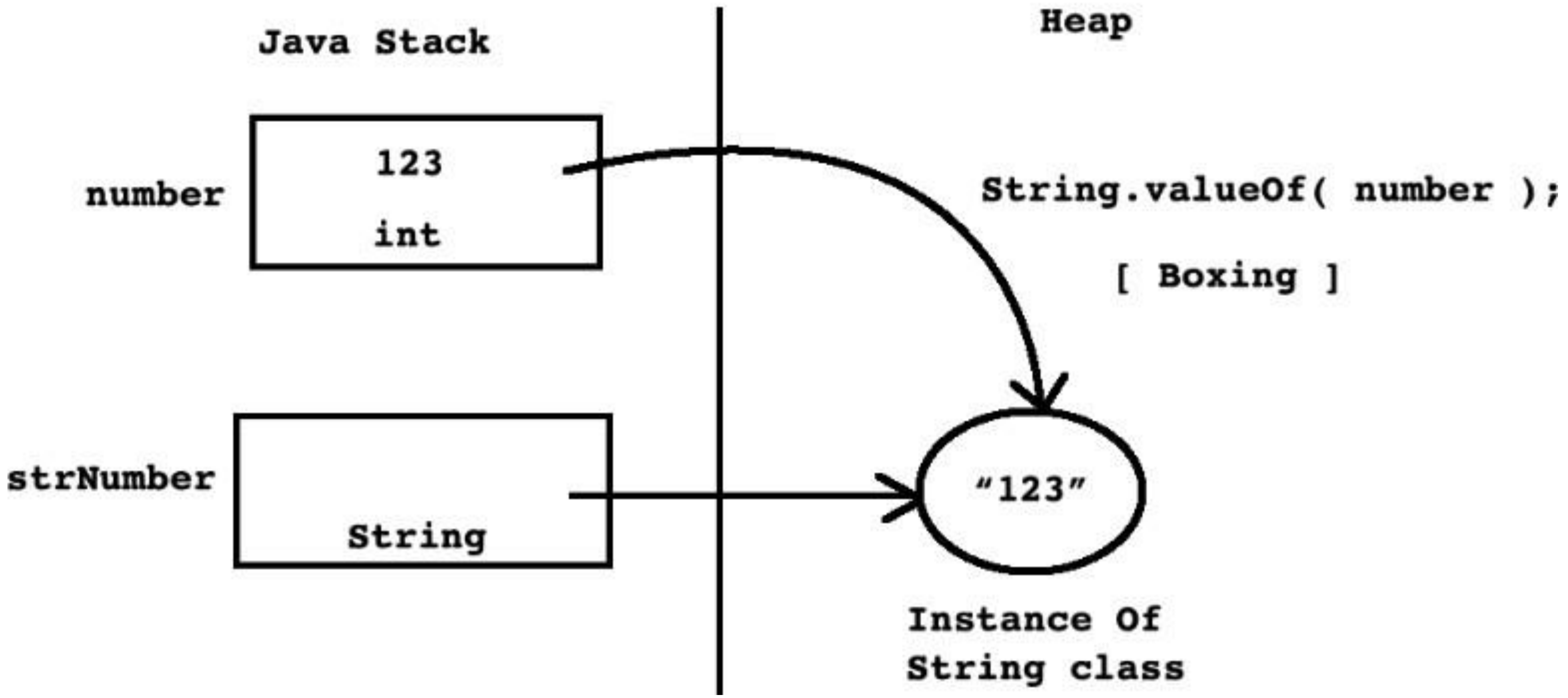


```
public static void main(String[] args) {
    int number = 123;
    //String str = Integer.toString( number );     //Boxing : OK
    String str = String.valueOf(number);          //Boxing : OK
    System.out.println("Str :    "+str);
}
```

- int n1=10; float f=3.5f; double d1=3.45
- String str1=String.valueOf(n1);
- String str2=String.valueOf(f);
- String str3=String.valueOf(d1);

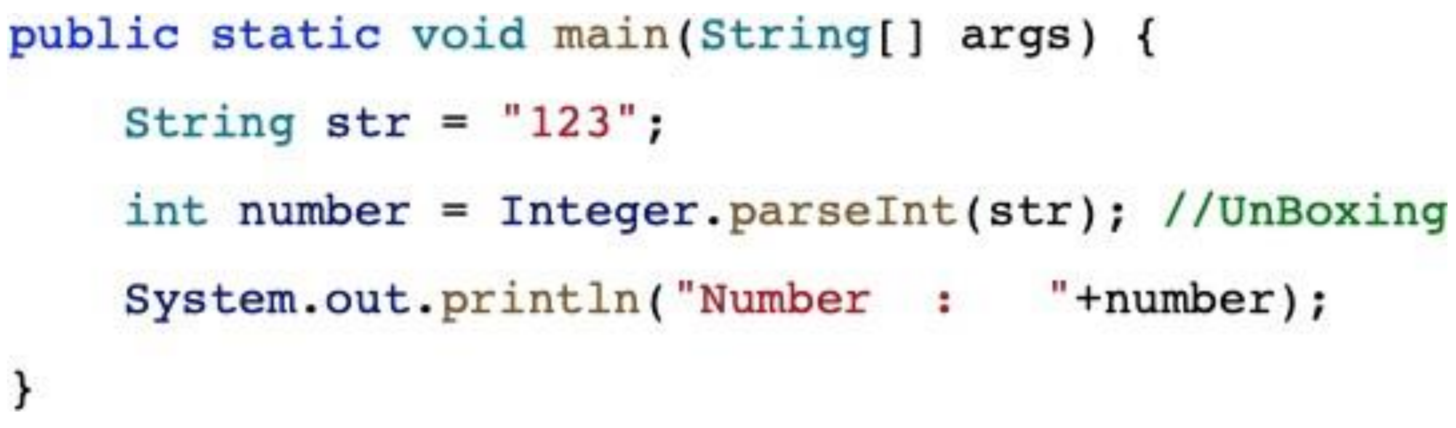
Boxing

```
int number = 123;
String strNumber = String.valueOf( number ); //Boxing
```



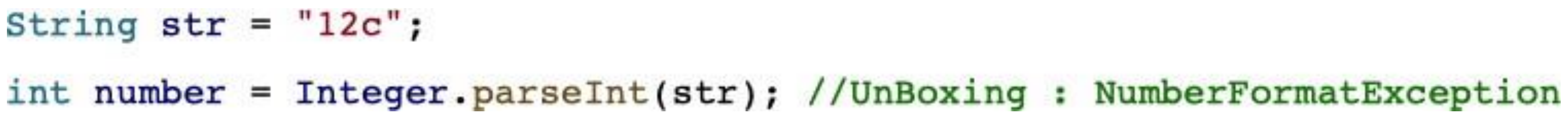
Unboxing

- Process of converting value of variable of non primitive type into primitive type is called unboxing.



```
public static void main(String[] args) {
    String str = "123";
    int number = Integer.parseInt(str); //UnBoxing
    System.out.println("Number   :    "+number);
}
```

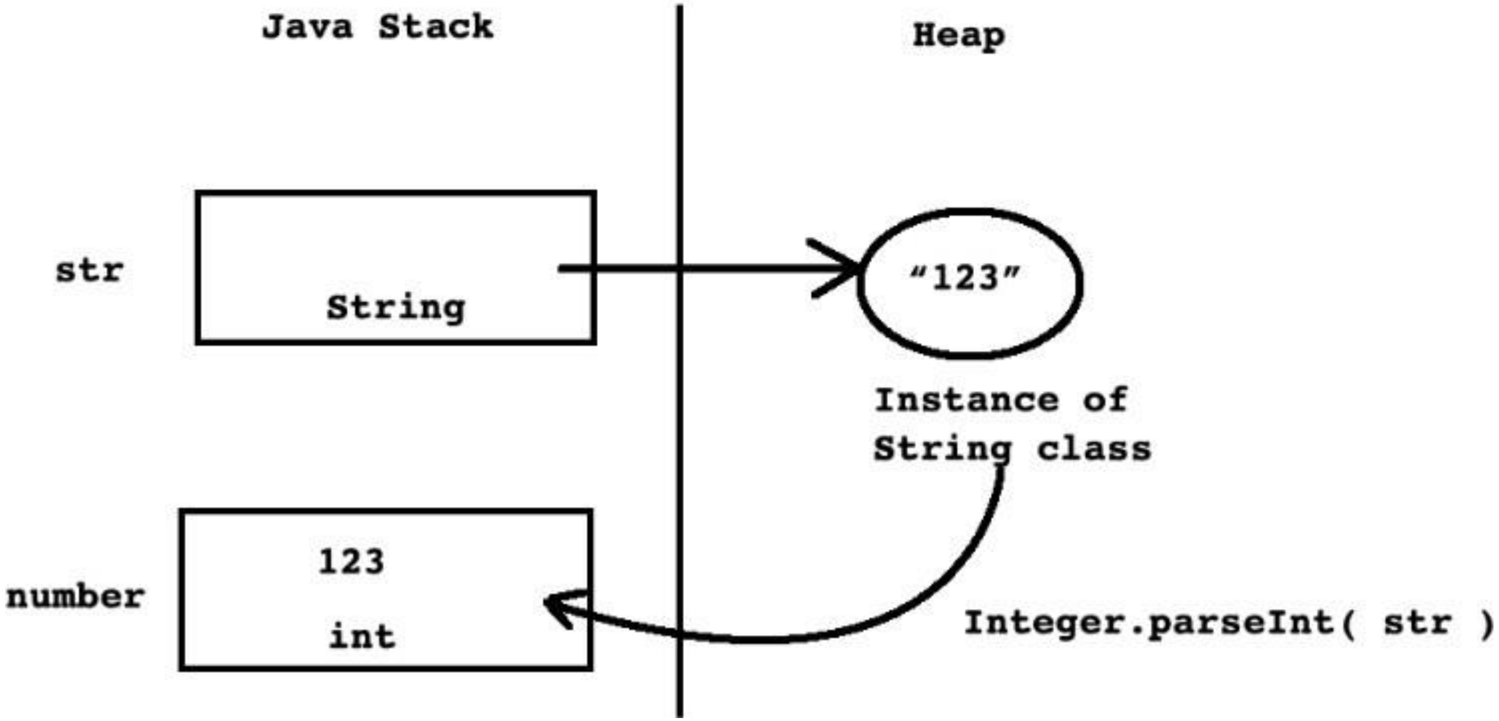
- If string does not contain parseable numeric value then **parseXXX( )** method throws **NumberFormatException.**



```
String str = "12c";
int number = Integer.parseInt(str); //UnBoxing : NumberFormatException
```

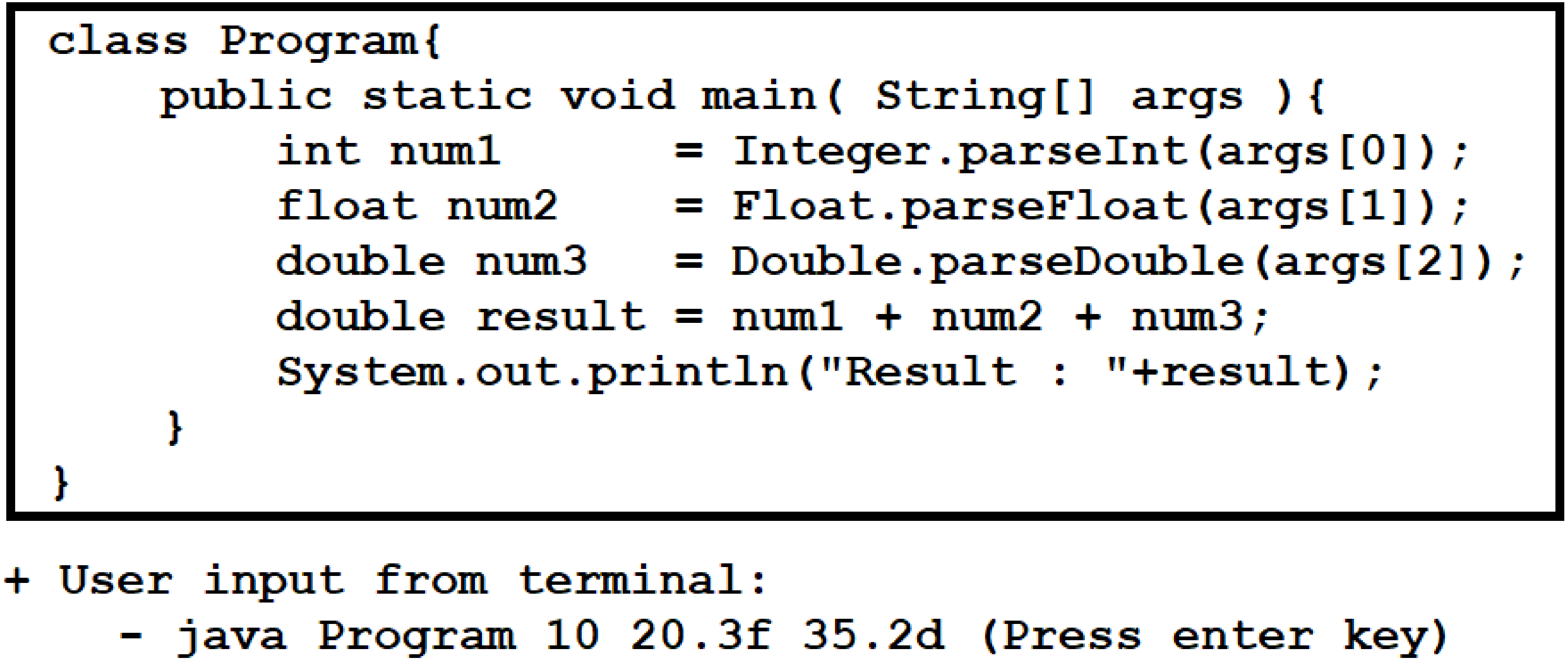
Unboxing

```
String str = “123”;
int number = Integer.parseInt( str ); //UnBoxing
```



Note : In case of boxing and unboxing one variable is primitive and other Is not primitive

Command line argument



```
class Program{
    public static void main( String[] args ){
        int num1      = Integer.parseInt(args[0]);
        float num2    = Float.parseFloat(args[1]);
        double num3   = Double.parseDouble(args[2]);
        double result = num1 + num2 + num3;
        System.out.println("Result : "+result);
    }
}
```

+ User input from terminal:
    - java Program 10 20.3f 35.2d (Press enter key)

Operators

- Arithmetic Operators
- Unary Operators
- Assignment Operator
- Relational Operators
- Logical Operators
- Ternary Operator
- Bitwise Operators
- Shift Operators

Operators Cont..

- **Arithmetic Operators**
  - They are used to perform simple arithmetic operations on primitive data types.

  * : Multiplication

  / : Division

  % : Modulo

  + : Addition

  – : Subtraction
- **Unary Operators**
  - Unary operators need only one operand. They are used to increment, decrement or negate a value.
  - Unary minus, used for negating the values.
  - eg : int a=20; int b=-a;
  - ++ :Increment operator, used for incrementing the value by 1. There are two varieties of increment operator.
  - Post-Increment : Value is first used for computing the result and then incremented.
  - Pre-Increment : Value is incremented first and then result is computed.
  - -- : Decrement operator, used for decrementing the value by 1. There are two varieties of decrement operator.
  - Post-decrement : Value is first used for computing the result and then decremented.
  - Pre-Decrement : Value is decremented first and then result is computed.
  - ! : Logical not operator, used for inverting a boolean value.
  - eg :   boolean jobDone=true;   boolean flag=!jobDone; System.out.println(flag);

Operators Cont..

- **Assignment Operators**
  - ‘=’ Assignment operator is used to assign a value to any variable. It has a right to left associativity.
  - Eg. int val = 500;
  - assignment operator can be combined with other operators to build a shorter version of statement called Compound Statement.
  - +=, for adding left operand with right operand and then assigning it to variable on the left.
  - -=, for subtracting left operand with right operand and then assigning it to variable on the left.
  - *=, for multiplying left operand with right operand and then assigning it to variable on the left.
  - /=, for dividing left operand with right operand and then assigning it to variable on the left.
  - %=, for assigning modulo of left operand with right operand and then assigning it to variable on the left.
- **Relational Operators**
  - These operators are used to check for relations like equality, greater than, less than. They return boolean result after the comparison and are used in looping statements and conditional if else statements.
  - ==, Equal to : returns true if left hand side is equal to right hand side.
  - !=, Not Equal to : returns true if left hand side is not equal to right hand side.
  - <, less than : returns true if left hand side is less than right hand side.
  - <=, less than or equal to : returns true if left hand side is less than or equal to right hand side.
  - >, Greater than : returns true if left hand side is greater than right hand side.
  - >=, Greater than or equal to: returns true if left hand side is greater than or equal to right hand side.

Operator Cont…

- **Logical Operators :**
  - These operators are used to perform “logical AND” and “logical OR” operation, i.e. the function similar to AND gate and OR gate in digital electronics.

    &&, Logical AND : returns true when both conditions are true.

    ||, Logical OR : returns true if at least one condition is true.
  - eg : int data=100;

```
  int data2=50;
 if(data > 60 && data2 < 100)
              System.out.println("test performed...");
else
              System.out.println("test not performed...");
```

- **Ternary operator :** Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary.
  - General format is :

    condition ? if true : if false

    The above statement means that if the condition evaluates to true, then execute the statements after the ‘?’ else execute the statements after the ‘:’.

    eg : int data=100;

    System.out.println(data>100?"Yes":"No");

Operators Cont..

- **Bitwise Operators :**
  - These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of Binary indexed tree.
  - &, Bitwise AND operator: returns bit by bit AND of input values.
  - |, Bitwise OR operator: returns bit by bit OR of input values.
  - ^, Bitwise XOR operator: returns bit by bit XOR of input values.
  - ~, Bitwise Complement Operator: This is a unary operator which returns the one’s compliment representation of the input value, i.e. with all bits inversed.
  - Eg :

    ```
    String binary[] = {  "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
                           "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111“  };
    int a = 3; // 0 + 2 + 1 or 0011 in binary
    int b = 6; // 4 + 2 + 0 or 0110 in binary
    int c = a | b;
    int d = a & b;
    int e = a ^ b;
    System.out.println("       a = " + binary[a]);
    System.out.println("       b = " + binary[b]);
    System.out.println("     a|b = " + binary[c]);
    System.out.println("     a&b = " + binary[d]);
    System.out.println("     a^b = " + binary[e]);       }
    ```

Operators Cont…

- **Shift Operators :**
  - These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively. They can be used when we have to multiply or divide a number by two.

  - <<, Left shift operator: shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.

  eg :

```
int a = 25;
System.out.println(a<<4); //25 * 16 = 400
a=-25;
System.out.println(a<<4);//-25 * 16 = -400
```

  - Signed right shift operator : The signed right shift operator '>>' uses the sign bit to fill the trailing positions. For example, if the number is positive then 0 will be used to fill the trailing positions and if the number is negative then 1 will be used to fill the trailing positions.

Example Shift Operations

- Assume if a = 60 and b = -60; now in binary format, they will be as follows −

- a = 0000 0000 0000 0000 0000 0000 0011 1100
- b = 1111 1111 1111 1111 1111 1111 1100 0100
- In Java, negative numbers are stored as 2's complement.

- Thus a >> 1 = 0000 0000 0000 0000 0000 0000 0001 1110
- And b >> 1 = 1111 1111 1111 1111 1111 1111 1110 0010

- Unsigned right shift operator
- The unsigned right shift operator '>>' do not use the sign bit to fill the trailing positions. It always fills the trailing positions by 0s.

- Thus a >>> 1 = 0000 0000 0000 0000 0000 0000 0001 1110
- And b >>> 1 = 0111 1111 1111 1111 1111 1111 1110 0010

Flow of Control

- Java executes one statement after the other in the order they are written
- Many Java statements are flow control statements: Conditional Stmt: if, if else, switch

Looping: for, while, do while

Escapes: break, continue, return

if Statement – different syntax options

```
   if
(expression)
    statement;
```

⇨ A single statement.

```
   if
(expression)
   {
    statements;
   }
```

⇨ A block of statements.

```
   if
(expression)
    statement;
   else
   statement;
```

⇨ Single statement in the if and a single statement in the else.

```
   if
(expression)
        statement;
   else
   {   statements;
   }
```

⇨ A single statement in the if and a block of statements in the else.

Conditional Operator

- The operator “ ? : ” is the only operator that takes three operands, each of which is an expression.
- The value of the whole expression equals the value of expr2 if expr1 is true, or equals the value of expr3 if expr1 is false.
- Syntax:

```
expr1 ? expr2 : expr3
```

switch Statement

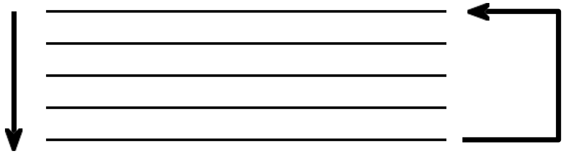
- The nested if can become complicated and unreadable.
- The switch statement is an alternative to the nested if.
- Syntax:

```
    switch(expression)
{
    case constant expr:
            statement(s);
            [break;]
    case constant expr:
            statement(s);
            [break;]
    case constant expr:
            statement(s);
            [break;]
    default :
            statement(s);
            [break;]
}
```

- Usually, but not always, the last statement of a case is break.
- default case is optional.

Loops

- Loops break the serial execution of the program.
- A group of statements is executed a number of times.



There are three kinds of loops :

- **while**
- **for**
- **do … while**

While – Loop

- Syntax:

```
while (expression)
   Statement;
```

or

```
while (expression)
{Statements;}
```

- The loop continues to iterate as long as the value of expression is true (expression differs from zero).

- Expression is evaluated each time before the loop body is executed.

- The braces { } are used to group declarations and statements together into a compound statement or block, so they are syntactically equivalent to a single statement.

for - Loop

- Syntax:

```
for (expr1 ; expr2 ; expr3)
      statement;
```

or

```
for (expr1 ; expr2 ; expr3)
{
    statements;
}
```

- Is equivalent to:

```
expr1;
while (expr2)
{
    {statements;}
    expr3;
}
```

do while Loop

- Syntax:

```
do
{
   Statements;
}while (expression);
```

- The condition expression for looping is evaluated only after the loop body had executed.

break Statement

- We have seen how to use the break statement within the switch statement.

- A break statement causes an exit from the innermost containing while, do, for or switch statement.

continue Statement

- In some situations, you might want to skip to the next iteration of a loop without finishing the current iteration.

- The **`continue`** statement allows you to do that.

- When encountered, **`continue`** skips over the remaining statements of the loop, but **`continues`** to the next iteration of the loop.

What is Scanner ?

- A class (java.util.Scanner) that represents text based parser(has inherent small ~ 1K buffer)
- It can parse text data from any source --Console input,Text file , socket, string

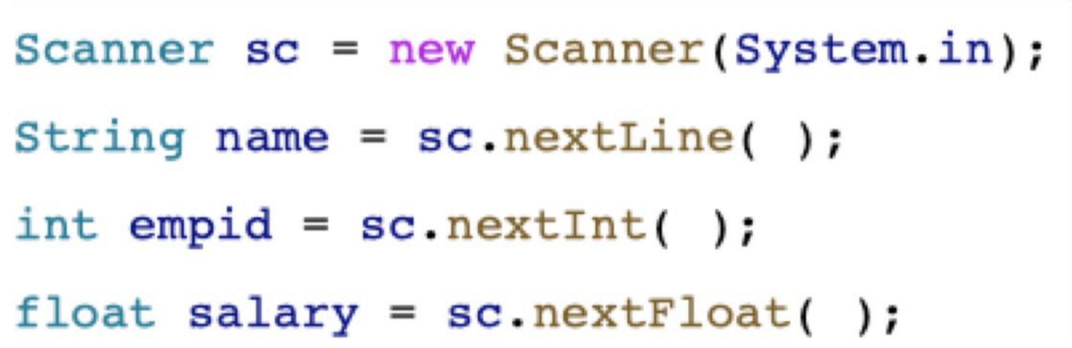
```
e.g.     Scanner input = new Scanner(System.in);
     System.out.print("Enter your name: ");
     String name = input.next ();
     System.out.println(“Your name is " + name);
     input.close();
```

User Input Using Scanner class.

- Scanner is a final class declared in java.util package.
- Methods of Scanner class:

```
1. public String nextLine()
2. public int nextInt()
3. public float nextFloat()
4. public double nextDouble()
```

- How to user Scanner?



```
Scanner sc = new Scanner(System.in);
String name = sc.nextLine( );
int empid = sc.nextInt( );
float salary = sc.nextFloat( );
```

Thank you.

akshita.Chanchlani@sunbeaminfo.com