

Object Oriented Programming with Java

Sun Microsystems

- Sun Microsystems was an American multinational technology company that was founded in 1982 and headquartered in California.
- It was founded by Vinod Khosla, Andy Bechtolsheim, and Scott McNealy, who were all Stanford University graduates. The company's name was derived from Stanford University Network (SUN), where the founders had met.
- Sun Microsystems was a pioneer in the development of Unix-based workstations, servers, and software.
- One of Sun Microsystems' most significant contributions to the technology industry was the development of the Java programming language.
- Sun Microsystems developed the Solaris operating system, which was based on Unix and was used in its workstations and servers.
- Over the years, Sun Microsystems made several acquisitions. Notable acquisitions included MySQL, an open-source database management system.
- In 2010, Oracle Corporation acquired Sun Microsystems for approximately \$7.4 billion.
- Following the acquisition by Oracle, the Sun Microsystems brand was phased out, and its products were rebranded under the Oracle name.

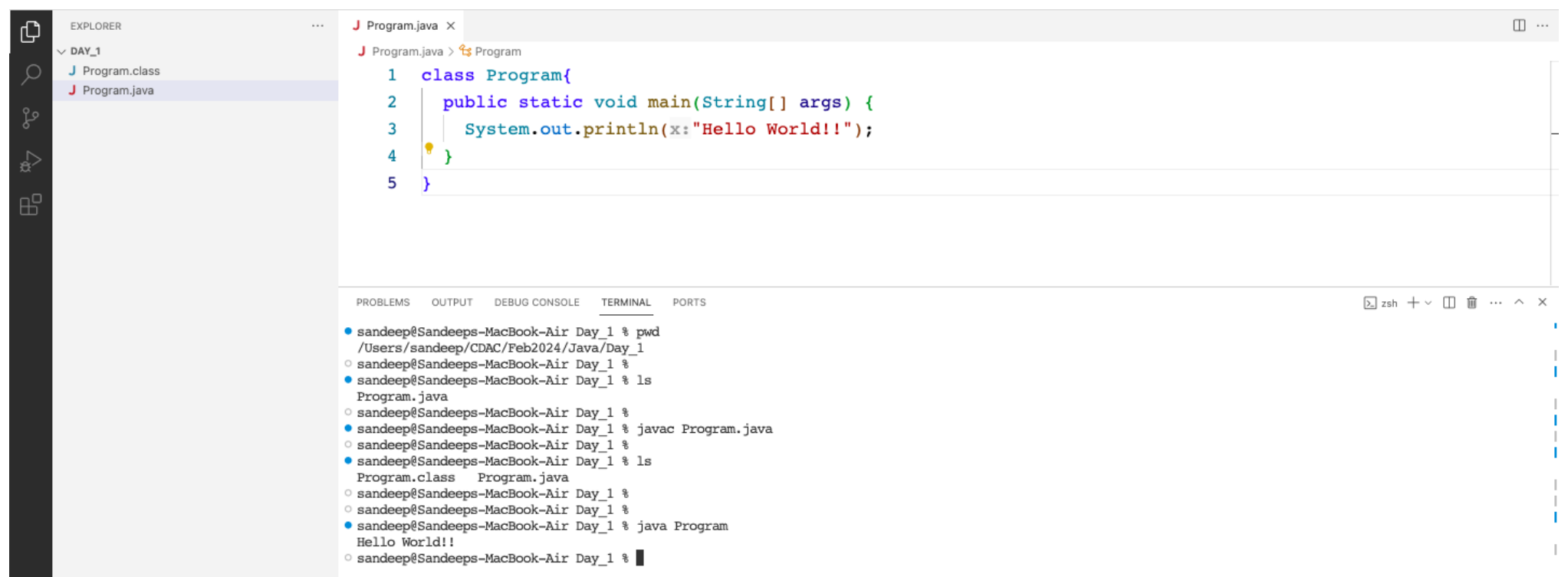
Java Introduction

- Java was developed by a team of engineers at Sun Microsystems, led by **James Gosling**, and was released in May 1995.
- The initial name of the language was Oak, but due to trademark issues, it was renamed to Java.
- The name "Java" was chosen as a reference to the coffee beans that were popular in the region where the language was developed.
- It is a high-level, class-based, object-oriented programming language that is derived from C/C++.
- It is a statically as well as strongly typed language.
- The Java Community Process (**JCP**) is responsible for standardizing Java. (URL: <https://www.jcp.org/en/home/index>)
- The slogan of the Java programming language is **"Write Once, Run Anywhere."**
- In 2010, Oracle Corporation acquired Sun Microsystems. Hence, it is now a product of Oracle.
- Java Programming language is both technology as well platform.
- JLS stands for Java Language Specification. It is a document that defines the syntax, semantics, and rules of the Java programming language. The JLS is maintained by Oracle Corporation.
- Reference: <https://docs.oracle.com/javase/specs/>

Description of Java Conceptual Diagram

- URL: <https://www.oracle.com/java/technologies/platform-glance.html>

Simple Hello Application



The screenshot shows an IDE with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a project named 'DAY_1' containing 'Program.class' and 'Program.java'. The code editor displays the following Java code:

```
1 class Program{
2     public static void main(String[] args) {
3         System.out.println(x:"Hello World!!");
4     }
5 }
```

The terminal at the bottom shows the execution of the program:

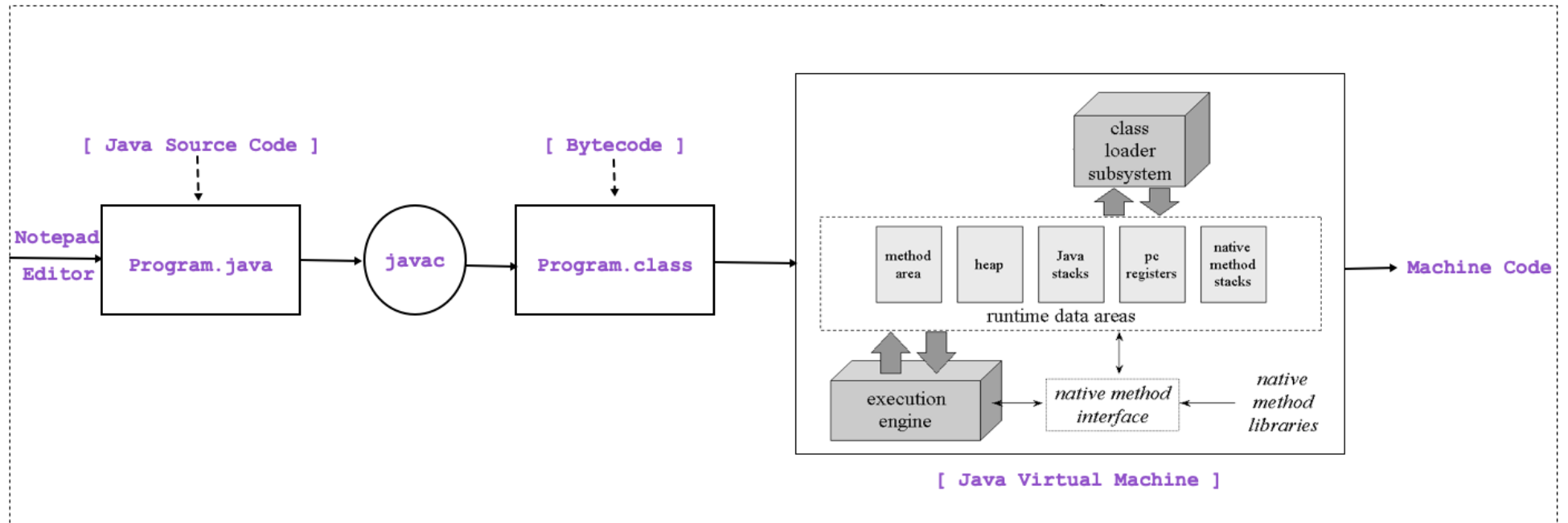
```
sandeep@Sandeeps-MacBook-Air Day_1 % pwd
/Users/sandeep/CDAC/Feb2024/Java/Day_1
sandeep@Sandeeps-MacBook-Air Day_1 % ls
Program.java
sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
sandeep@Sandeeps-MacBook-Air Day_1 % ls
Program.class Program.java
sandeep@Sandeeps-MacBook-Air Day_1 % java Program
Hello World!!
sandeep@Sandeeps-MacBook-Air Day_1 %
```

- .java file is text file which contains Java source code.
- javac is a java compiler. It compiles java source code and convert into bytecode.
- .class file is a binary file which contains bytecode.

JVM Architecture

Object Oriented Programming with Java

- Consider below Java flow of execution.



- Components of Java Virtual Machine:
 - Class loader subsystem
 - Bootstrap Class Loader
 - Extension Class Loader
 - System Class Loader
 - Custom ClassLoader
 - Runtime data areas
 - Method Area
 - Heap
 - Java Stacks
 - PC Registers
 - Native Method Stacks
 - Execution engine
 - Interpreter
 - Just-In-Time (JIT) Compiler
 - Garbage Collector

Points to remember

- javac is a Java compiler which read java source code and convert it into byte code. It keeps bytecode in .class file.
- .class file gets created per type(interface/class/enum) defined inside .java file. Hence name of source file and .class file need not to be same.
- If .java file is empty then Java compiler do not generate .class file.
- Bytecode in Java is not a machine code rather it is a object oriented assembly language code designed for JVM.
- If we want to disassemble the bytecode then we should javap tool.
- Example:

```
javac Program.java      //Output: Program.class
javap -c Program.class  //Output: Bytecode
```

Modifiers in Java

- In Java, modifiers are keywords that are used to change the behavior of classes, methods, variables, and other elements of a Java program.
- There are 12 modifiers in Java.
 - private
 - protected
 - public
 - static
 - final
 - abstract
 - interface
 - transient
 - synchronized
 - volatile
 - strictfp
 - native

Access Modifiers in Java

- There are 4 access modifiers in Java which is used to control visibility of the members declared inside Type(interface/class/enum).
 - private
 - package level private(also called as default)

- protected
- public

	Same Package			Different Package	
Access Modifier	Same Class	Sub Class	Non Sub Class	Sub Class	Non Sub Class
private	A	NA	NA	NA	NA
package level private	A	A	A	NA	NA
protected	A	A	A	A	NA
public	A	A	A	A	A

Java Virtual Machine Threads

- When a Java application starts, several threads are automatically created and started by the JVM to handle various tasks and provide runtime services.
 - **Main Thread**
 - It executes the main() method .
 - **Reference Handler Thread**
 - It is responsible for handling reference objects that are enqueued by the garbage collector.
 - **Finalizer Thread**
 - It is responsible for running the finalize() method of objects that are eligible for finalization.
 - **Signal Dispatcher Thread**
 - is responsible for handling operating system signals, such as SIGINT (interrupt signal) and SIGTERM (termination signal).
 - **Compiler Threads**
 - The JVM may create and start one or more compiler threads to compile Java bytecode into native machine code.
 - **Garbage Collector Threads**
 - To perform garbage collection.
 - **Other System Threads**
- jstack is command line tool which is used to take thread dump.
 - First get the process id:
 - jcmd //or
 - ps -ef | grep java
 - Use process id with jstack
 - jstack pid

Entry point method in Java

- main method is considered as entry point method in Java.
- Legal main method signatures:
 - public static void main(String[] args)
 - public static void main(String args[])
 - public static void main(String... args)
- As discussed above, when we start execution of Java application, JVM create and start execution of main thread and this main thread is responsible for invoking/calling main method.
- Java compiler do not check whether main method is defined in a class or not? It also doesn't check whether signature of main method is valid of invalid. It's a job of JVM. Hence we compile empty .java file, we dont get any error.
- Consider below code:

```
//Program.java
class Program{
    //Don't define anything
}
```

```
//Compile code
javac Program.java //Program.class -> No Compiler Error
```

```
java Program //Error: Main method not found in class Main
```

- We can define main method per class. But only one main method will be considered as entry point method. Consider below code:

Object Oriented Programming with Java

```
//Program.java
class A{
    public static void main( String[] args ){
        System.out.println("A.main");
    }
}
class B{
    public static void main( String[] args ){
        System.out.println("B.main");
    }
}
class Program{
    public static void main( String[] args ){
        System.out.println("Program.main");
    }
}
```

```
javac Program.java //A.class, B.class, Program.class
java A //A.main
java B //B.main
java Program //Program.main
```

- We can define multiple main methods with different signature inside class. Consider below code:

```
class Program{
    public static void main( ){
        System.out.println("Hello from main()");
    }
    public static void main( String[] args ){
        Program.main( ); //Static methods are designed to call on class name. It's not recursive call.
    }
}
```

- Reference: <https://www.baeldung.com/java-main-method>

Why main method is static?

- Non static methods are designed to call on instance and static methods are designed to called on class name.
- Let us understand what if main method is non static in java.
- Scenario 1:

```
abstract class Program{
    public void main( String[] args ){ //non static main method
        //TODO
    }
}
```

- We can not create instance of abstract class. Here JVM will fail.
- Scenario 2:

```
class Program{
    private Program( ){ //private constructor
        //TODO
    }
    public void main( String[] args ){ //non static main method
        //TODO
    }
}
```

- If constructor is private then we can not instantiate it in different class. Here JVM will fail
- Scenario 3:

```
class Program{
    public Program( int arg1, float arg2, double arg3, String arg4, Date arg5,...){ //public constructor
        //TODO
    }
}
```

Object Oriented Programming with Java

```
}  
public void main( String[] args ){ //non static main method  
    //TODO  
}  
}
```

- Here instance must be created by passing arguments which is increasing JVM headache.
- To overcome similar problems, JVM specification says main method must be static.
- **Note:** This question is for interview preparation only. To get full understanding you must first understand constructor, abstract class, concrete class etc. We will discuss it after some time.

Reading Assignment

- Unnamed Classes and Instance Main Methods
- Reference: <https://docs.oracle.com/en/java/javase/21/language/unnamed-classes-and-instance-main-methods.html>

More About src.zip, rt.jar and java docs

- Java home directory contains src.zip file. It contains source code of Java API. It proves that Java is open source technology.
- Java home/jre directory contains rt.jar file. It contains compiled code of Java API. To run any Java application, JVM need it.
- Java docs contain help/documentation of Java API in the form of html pages. It helps Java application developer to use classes and methods.

What is the meaning of System.out.println

- Consider the definition of System class:

```
package java.lang;  
import java.io.*;  
public final class System {  
    public final static InputStream in = null;  
    public final static PrintStream out = null;  
    public final static PrintStream err = null;  
  
    public static void exit(int status) {  
        Runtime.getRuntime().exit(status);  
    }  
    public static void gc() {  
        Runtime.getRuntime().gc();  
    }  
}
```

- Check the documentation of System class:
 - <https://docs.oracle.com/javase/8/docs/api/java/lang/System.html>
- **System is a final class declared in java.lang package.**
- in, out and err are static reference fields of System class. Hence we can access it like:
 - System.in
 - System.out
 - System.err
- **out is reference variable of java.io.PrintStream class. It is declared as public static final field inside System class.**
- Consider the definition of PrintStream class:

```
package java.io;  
public class PrintStream extends FilterOutputStream implements Appendable, Closeable{  
    //Consider overloaded print methods  
    public void print(boolean);  
    public void print(char){}  
    public void print(int){}  
    public void print(long){}  
    public void print(float){}  
    public void print(double){}  
    public void print(char[]){}  
    public void print(String){}  
    public void print(Object){}  
  
    //Consider overloaded println methods  
    public void println(){}  
    public void println(boolean){}
```

```
public void println(char){}
public void println(int){}
public void println(long){}
public void println(float){}
public void println(double){}
public void println(char[]){}
public void println(String){}
public void println(Object){}

//Consider overloaded printf methods
public java.io.PrintStream printf(String, Object...){}
public java.io.PrintStream printf(Locale, String, Object...){}
}
```

- **println() is non static overloaded method of java.io.PrintStream class.**
- Check the documentation of PrintStream class:
 - <https://docs.oracle.com/javase/8/docs/api/java/io/PrintStream.html>

print versus println versus printf

- print is non static overloaded method of java.io.PrintStream class. It is used to write data on terminal / file etc.
- print method write data to destination but keep cursor on same line.

```
public static void main( String[] args ){
    System.out.print("Hello ");
    System.out.print("World");
    System.out.print("!!");
    System.out.print("\n");
}
```

- Output will be

```
Hello World!!
```

- println is non static overloaded method of java.io.PrintStream class. It is used to write data on terminal / file etc.
- println method write data to destination but move cursor to the next line.

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

- Output will be

```
Hello
World
!!
```

- printf is non static overloaded method of java.io.PrintStream class. It is used to write data on terminal / file etc.
- printf method is used to write formatted data to destination. It is similar to printf function in C/C++.

```
public static void main( String[] args ){
    String name = "Sandeep Kulange";
    int empid = 3778;
    float salary = 34700.50f;
    System.out.printf("%s%d%f\n", name, empid, salary);
    //or
    System.out.printf("%-20s%-5d%-10.2f\n", name, empid, salary);
}
```


Object Oriented Programming with Java

- **Note:** Return type of print/println method is void but Return type of printf method is PrintStream which is class.

How to write "Hello World!!" program without giving semicolon?

- Consider below code:

```
class Program{
    public static void main( String[] args ){
        if( System.out.printf("Hello World") != null ){
        }
    }
}
```

- In java, null is a literal which is used to initialize reference / non primitive type of variable.
- if statement is control flow statement which is used for decision making.
- "!=" is relational operator which results boolean value(true / false).

Data Types

- In programming, we can not do coding without knowing the data types. So lets first understand what is data type?
- Most of the time, the term "data type" is simply referred to as "type". Therefore, we will use the terms "data type" and "type" interchangeably.
- Data type of any variable describes following properties:
 - **Memory Allocation:**
 - Different data types require different amounts of memory and have different memory layouts.
 - So data type define, how much memory is required to store the data.
 - **Type of Data:**
 - Data types define what kind of data can be stored in a variable.
 - For example, an integer data type can only store whole numbers, while a floating-point data type can store decimal numbers. Other data types include strings (text), booleans (true/false), and arrays (collections of values).
 - **Operations:**
 - Each data type has a set of operations that can be performed on it.
 - So data type define, which operations can be performed on the data stored in memory.
 - **Range of data:**
 - Data types also define the range of values that can be stored in a variable.
 - For example, an integer data type might have a range from -2,147,483,648 to 2,147,483,647, while a floating-point data type might have a range from -1.7976931348623157e+308 to 1.7976931348623157e+308.

Statically typed language versus Dynamically typed language

- Statically typed languages perform type checking at compile time whereas dynamically typed languages perform type checking at runtime.
- Statically-typed languages require us to declare the data types of our variables before we use them, while dynamically-typed languages do not.
- In static typing, types are associated with variables not values. In dynamic typing, types are associated with values not variables.
- Static typing allows many type errors to be caught early in the development cycle. Dynamic typing type allows errors to be caught at runtime.
- C, C++, Java, C# etc are statically typed languages and JavaScript, Python, Ruby, PHP, Groovy are dynamically typed languages.
- Consider the Java example:

```
int num;
num = 5;
```

- Consider the Groovy example:

```
num=5
```

Classification of Data Types

- Java is statically as well as strongly typed language.
- Types of data types in Java:
 1. Primitive data types(also called as value types)
 2. Non Primitive data types (also called as reference types)

Primitive data types

Sr.No.	Primitive Type	Size(In Bytes)	Default Value(For Fields)	Wrapper Class
1	boolean	Not defined	false	java.lang.Boolean
2	byte	1	0	java.lang.Byte
3	char	2	'\u0000'	java.lang.Character
4	short	2	0	java.lang.Short
5	int	4	0	java.lang.Integer
6	float	4	0.0f	java.lang.Float
7	double	8	0.0d	java.lang.Double
8	long	8	0L	java.lang.Long

- Reference: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

Non Primitive data types

- Below types are non primitive types in Java:
 - 1. Interface
 - 2. Class
 - 3. Enum
 - 4. Array
- Reference: <https://docs.oracle.com/javase/tutorial/reflect/class/index.html>

Wrapper classes

- In Java, primitive types are not classes. Hence we can not use new operator with it. Also we can not call method on it.

```
int number = new int( ); //Not OK
int number = new int( 10 ); //Not OK
int number = 10; //OK
String str = number.toString(); //Not OK
```

- In Java, we get class corresponding to each primitive type. Such class is called as wrapper class.

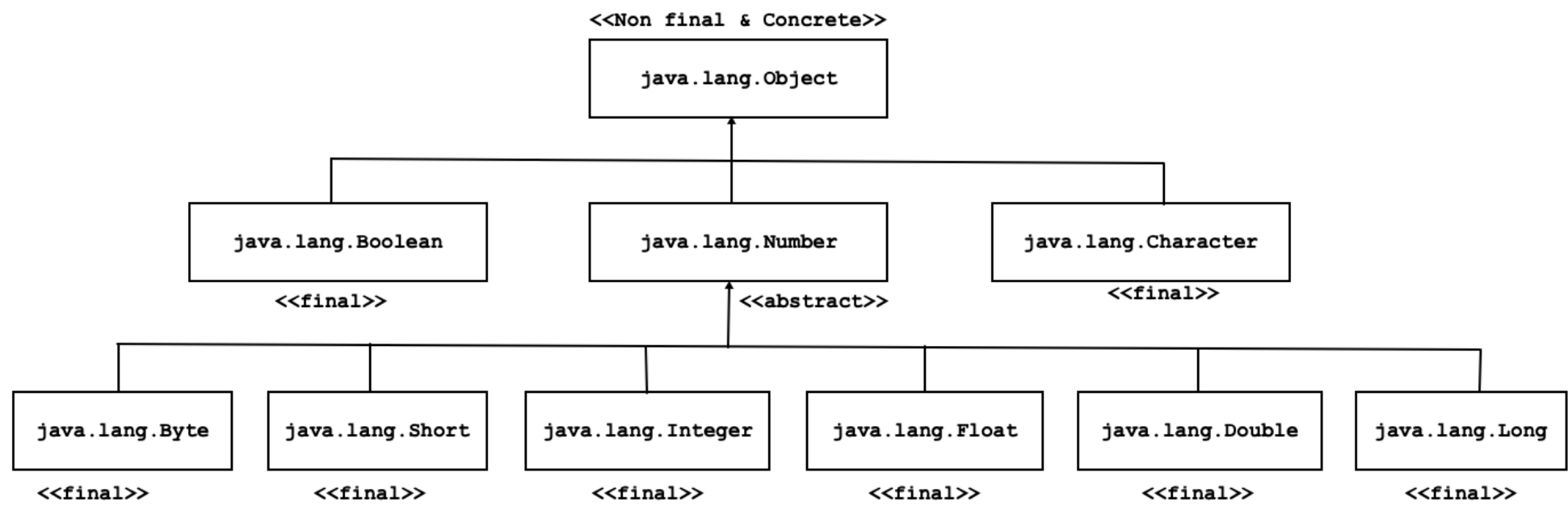
Sr.No.	Primitive Type	Wrapper Class	Parent / Super Class
1	boolean	java.lang.Boolean	java.lag.Object
2	byte	java.lang.Byte	java.lang.Number
3	char	java.lang.Character	java.lang.Object
4	short	java.lang.Short	java.lang.Number
5	int	java.lang.Integer	java.lang.Number
6	float	java.lang.Float	java.lang.Number
7	double	java.lang.Double	java.lang.Number
8	long	java.lang.Long	java.lang.Number

- We can create instance of Wrapper class also we can invoke methods on it.

```
Integer number = new Integer( 10 ); //OK
int value = number.intValue( ); //Ok
```


Object Oriented Programming with Java

- Wrapper class hierarchy:



- All wrapper classes are immutable as well as final and declared in java.lang package.

```
public class Program {
    public static void main(String[] args) {
        Integer n1 = new Integer(10);

        Integer n2 = n1 + 5;

        System.out.println("n1::"+n1); //10
        System.out.println("n2::"+n2); //15
    }
}
```

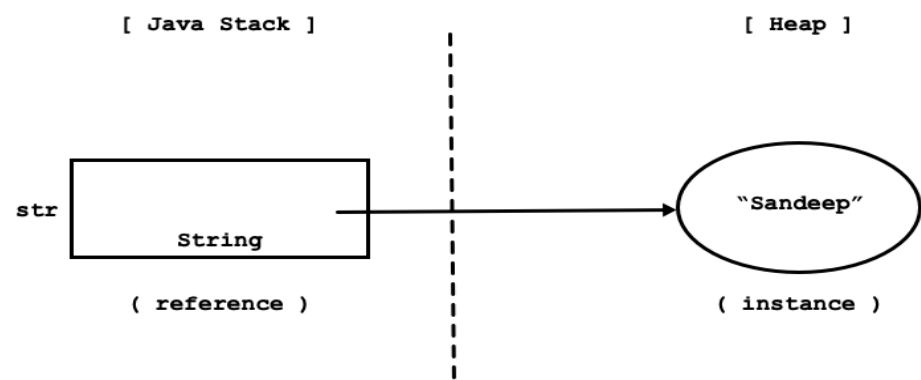
- Reference: <https://docs.oracle.com/javase/8/docs/api/>

String

- String is not a primitive type or built in type in Java. It's a final class declared in java.lang package. Since it's class, it is considered as non primitive type or reference type.
- We can create instance of String using new operator as well as without new operator.
- Consider String instance using new operator.

```
public static void main( String[] args ){
    String str = new String("Sandeep");
}
```

- Here str is called as object reference or simply reference.
 - "new String("Sandeep")" is called as instance(Object in C++).
- Consider memory representation:



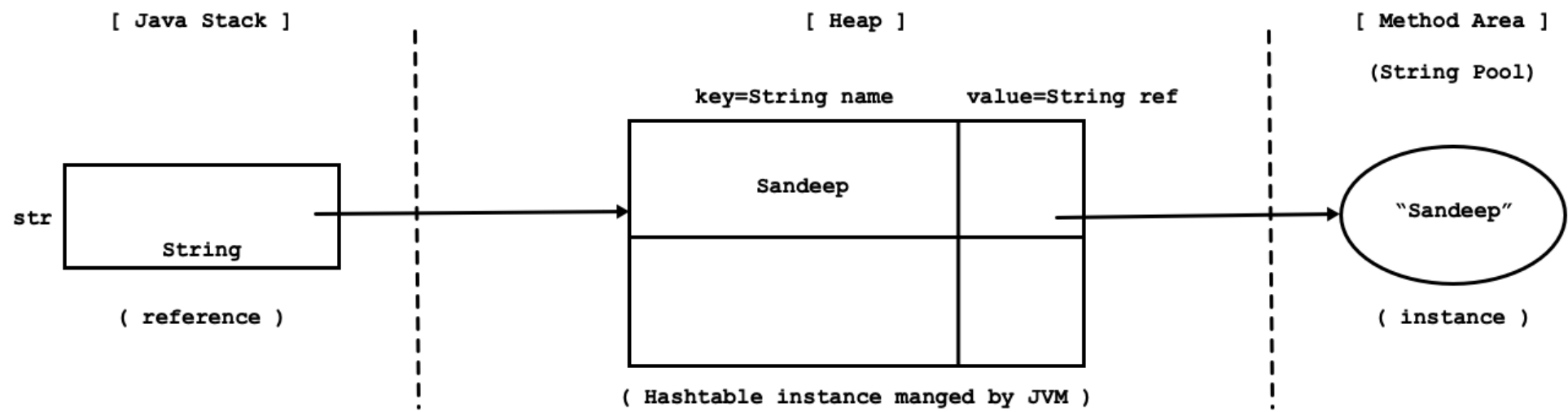
- Consider String instance without new operator.

```
public static void main( String[] args ){
    String str ="Sandeep";
}
```

- Here str is called as object reference or simply reference.
 - "Sandeep" is called as String constant / String literal.

Object Oriented Programming with Java

- Consider memory representation:



- Note:** We will discuss difference and between two and other string related stub in String handling.

Variables

- Definition:
 - An entity, whose value can be changed is called as variable.
 - Name given to memory location is called as variable.
- The Java programming language defines the following kinds of variables:
 - Local variable(Also called as method local variable in Java)
 - Parameter (Also called as method parameter in Java)
 - Instance variable(Non Static Field Declared inside class)
 - Class Variable(Static Field Declared inside class)
- Consider below code:

```
class Program{
    //Non static field
    int num1 = 10; //num1 is Instance Variable

    //Static Field
    static int num2 = 20; //num2 is Class Variable
    public static void main( String[] args ){ //args is method parameter
        int num3 = 30; //num3 is method local variable
        int num4 = 40; //num4 is method local variable
        Program.print( num4 ); //Here num4 is passed as method argument
    }
    private static void print( int num5 ){ //num5 is method parameter
        System.out.print( num5 );
    }
}
```

- Naming Convention:
 - Variable names are case-sensitive.
 - Example:

```
int num1; //OK;
int NUM1; //OK
```

- A variable's name can be any legal identifier — an unlimited-length sequence of letters and digits, beginning with a letter, the dollar sign "\$", or the underscore character "_".The convention, however, is to always begin your variable names with a letter, not "\$" or "_". Additionally, the dollar sign character, by convention, is never used at all. A similar convention exists for the underscore character; while it's technically legal to begin your variable's name with "_", this practice is discouraged.
- White space is not permitted.
- Subsequent characters may be letters, digits, dollar signs, or underscore characters.
- When choosing a name for your variables, use full words instead of cryptic abbreviations. Doing so will make your code easier to read and understand.
- Name of the keyword can not be used for variable.
 - Example:

```
Calendar calendar; //OK
Date date; //OK
```

Object Oriented Programming with Java

```
Package package; //Not Ok: package is a keyword
```

- Reference: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

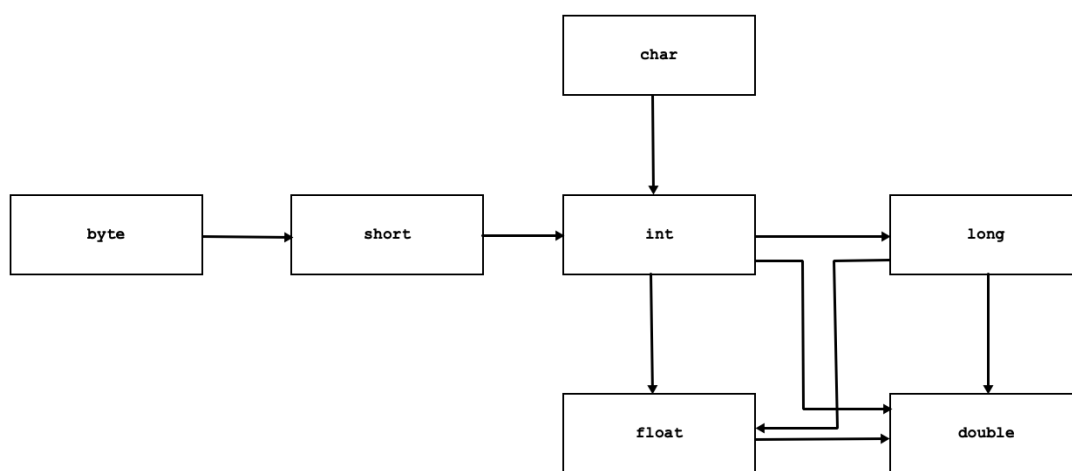
- If the name you choose consists of only one word, spell that word in all lowercase letters. If it consists of more than one word, capitalize the first letter of each subsequent word.
- Example:

```
number, gearRatio, currentGear, dateOfBirth, joinDate
```

- If your variable stores a constant value, such as `static final int NUM_GEAR = 6`, the convention changes slightly, capitalizing every letter and separating subsequent words with the underscore character. By convention, the underscore character is never used elsewhere.

- Reference: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html>

Widening



- Widening is the process of converting value of variable of narrower type into wider type.
- Consider below examples.
- byte to other types conversion:

```
public static void main(String[] args) {
    byte byteNumber = 125;
    //char character = byteNumber; //Not OK
    short shortNumber = byteNumber; //OK
    int integerNumber = byteNumber; //OK
    long longNumber = byteNumber; //OK
    float floatNumber = byteNumber; //OK
    double doubleNumber = byteNumber; //OK

    System.out.println("Byte:: "+byteNumber); //125
    System.out.println("Short:: "+shortNumber); //125
    System.out.println("Integer:: "+integerNumber); //125
    System.out.println("Long:: "+longNumber); //125
    System.out.println("Float:: "+floatNumber); //125.0
    System.out.println("Double:: "+doubleNumber); //125.0
}
```

- short to other types conversion:

```
public static void main(String[] args) {
    short shortNumber = 32767;
    //boolean booleanValue = shortNumber; //Not OK
    //byte byteNumber = shortNumber; //Not OK
    //char character = shortNumber; //Not OK
    int integerNumber = shortNumber; //OK
    long longNumber = shortNumber; //OK
    float floatNumber = shortNumber; //OK
    double doubleNumber = shortNumber; //OK

    System.out.println("Short:: "+shortNumber); //32767
    System.out.println("Integer:: "+integerNumber); //32767
    System.out.println("Long:: "+longNumber); //32767
    System.out.println("Float:: "+floatNumber); //32767.0
}
```

Object Oriented Programming with Java

```
System.out.println("Double:: "+doubleNumber);    //32767.0
}
```

- char to other types conversion:

```
public static void main(String[] args) {
    char character = 'A';    //Code point of A is 65

    //boolean booleanValue = character; //Not OK
    //byte byteNumber = character;    //Not OK
    //short shortNumber = character;    //Not OK
    int integerNumber = character;    //OK
    long longNumber = character;    //OK
    float floatNumber = character;    //OK
    double doubleNumber = character;    //OK

    System.out.println("Char:: "+character);    //A
    System.out.println("Integer:: "+integerNumber); //65
    System.out.println("Long:: "+longNumber);    //65
    System.out.println("Float:: "+floatNumber); //65.0
    System.out.println("Double:: "+doubleNumber);    //65.0
}
```

- int to other types conversion:

```
public static void main(String[] args) {
    int integerNumber = 2147383647;

    //boolean booleanValue = integerNumber; //Not OK
    //byte byteNumber = integerNumber;    //Not OK
    //short shortNumber = integerNumber;    //Not OK
    //char character = integerNumber;    //Not OK
    long longNumber = integerNumber;    //OK
    float floatNumber = integerNumber;    //OK
    double doubleNumber = integerNumber;    //OK

    System.out.println("Integer:: "+integerNumber); //2147383647
    System.out.println("Long:: "+longNumber);    //2147383647
    System.out.println("Float:: "+floatNumber); //2.14738368E9
    System.out.println("Double:: "+doubleNumber);    //2.14738368E9
}
```

- long to other types conversion:

```
public static void main(String[] args) {
    long longNumber = 9223372036854775807L;

    //boolean booleanValue = longNumber;    //Not OK
    //byte byteNumber = longNumber;    //Not OK
    //short shortNumber = longNumber;    //Not OK
    //char character = longNumber;    //Not OK
    //int integerNumber = longNumber;    //Not OK
    float floatNumber = longNumber;    //OK
    double doubleNumber = longNumber;    //OK

    System.out.println("Long:: "+longNumber);    //9223372036854775807
    System.out.println("Float:: "+floatNumber); //9.223372E18
    System.out.println("Double:: "+doubleNumber);    //9.223372036854776E18
}
```

- float to other types conversion:

```
public static void main(String[] args) {
    float floatNumber = 12345.230f;

    //boolean booleanValue = floatNumber;    //Not OK
    //byte byteNumber = floatNumber;    //Not OK
    //short shortNumber = floatNumber;    //Not OK
    //char character = floatNumber; //Not OK
}
```

Object Oriented Programming with Java

```
//int integerNumber = floatNumber; //Not OK
//long longNumber = floatNumber; //Not OK
double doubleNumber = floatNumber; //OK
System.out.println("Float:: "+floatNumber); //12345.23
System.out.println("Double:: "+doubleNumber); //12345.23046875
}
```

- double to other types conversion:

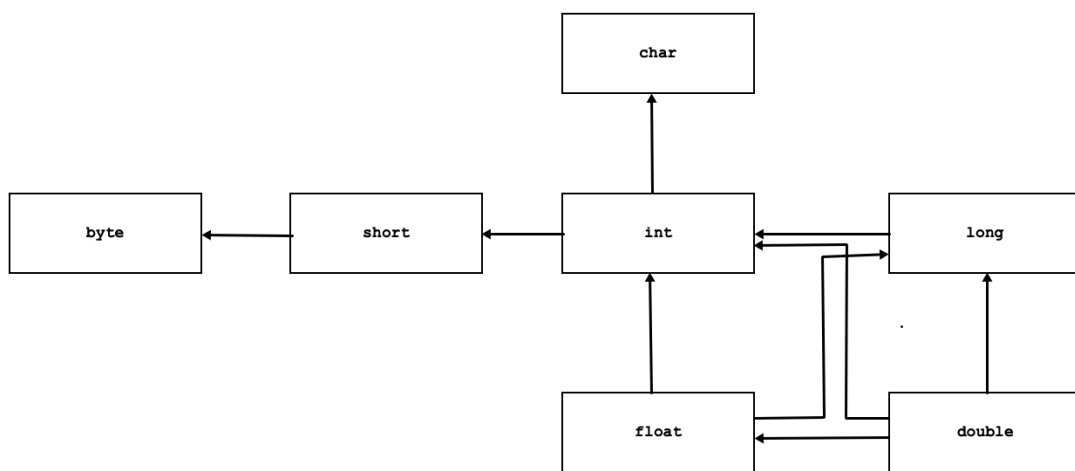
```
public static void main(String[] args) {
    double doubleNumber = 12345421.293847d;

    //boolean booleanValue = doubleNumber; //Not OK
    //byte byteNumber = doubleNumber; //Not OK
    //short shortNumber = doubleNumber; //Not OK
    //char character = doubleNumber; //Not OK
    //int integerNumber = doubleNumber; //Not OK
    //long longNumber = doubleNumber; //Not OK
    //float floatNumber = doubleNumber; //Not OK
    System.out.println("Double:: "+doubleNumber); //1.2345421293847E7
}
```

- During widening, explicit type casting is not required.

Narrowing

- Narrowing is the process of converting value of variable of wider type into narrower type.



- Consider below examples.
- double to other types conversion:

```
public static void main(String[] args) {
    double doubleNumber = 121.65d;

    byte byteNumber = ( byte )doubleNumber;
    char character = ( char )doubleNumber;
    short shortNumber = ( short )doubleNumber;
    int integerNumber = ( int )doubleNumber;
    long longNumber = ( long )doubleNumber;
    float floatNumber = ( float )doubleNumber;

    System.out.println("Byte Number:: "+byteNumber); //121
    System.out.println("Character Value:: "+character); //y
    System.out.println("Short Number:: "+shortNumber); //121
    System.out.println("Integer Number:: "+integerNumber); //121
    System.out.println("Long Number:: "+longNumber); //121
    System.out.println("Float Number:: "+floatNumber); //121.65
    System.out.println("Double Number:: "+doubleNumber); //121.65
}
```

- float to other types conversion:

```
public static void main(String[] args) {
    float floatNumber = 121.65f;

    byte byteNumber = ( byte )floatNumber;
    char character = ( char )floatNumber;
```

Object Oriented Programming with Java

```
short shortNumber = (short)floatNumber;
int integerNumber = (int)floatNumber;
long longNumber = (long) floatNumber;

System.out.println("Byte Number:: "+byteNumber);    //121
System.out.println("Character Value:: "+character); //y
System.out.println("Short Number:: "+shortNumber);  //121
System.out.println("Integer Number:: "+integerNumber); //121
System.out.println("Long Number:: "+longNumber);    //121
System.out.println("Float Number:: "+floatNumber);  //121.65
}
```

- long to other types conversion:

```
public static void main(String[] args) {
    long longNumber = 121;

    byte byteNumber = ( byte )longNumber;
    char character = ( char )longNumber;
    short shortNumber = (short)longNumber;
    int integerNumber = (int)longNumber;

    System.out.println("Byte Number:: "+byteNumber);    //121
    System.out.println("Character Value:: "+character); //y
    System.out.println("Short Number:: "+shortNumber);  //121
    System.out.println("Integer Number:: "+integerNumber); //121
    System.out.println("Long Number:: "+longNumber);    //121
}
```

- int to other types conversion:

```
public static void main(String[] args) {
    int integerNumber = 121;

    byte byteNumber = ( byte )integerNumber;
    char character = ( char )integerNumber;
    short shortNumber = (short)integerNumber;

    System.out.println("Byte Number:: "+byteNumber);    //121
    System.out.println("Character Value:: "+character); //y
    System.out.println("Short Number:: "+shortNumber);  //121
    System.out.println("Integer Number:: "+integerNumber); //121
}
```

- char to other types conversion:

```
public static void main(String[] args) {
    char character = 'y';

    byte byteNumber = ( byte )character;
    short shortNumber = (short)character;

    System.out.println("Byte Number:: "+byteNumber);    //121
    System.out.println("Character Value:: "+character); //y
    System.out.println("Short Number:: "+shortNumber);  //121
}
```

- short to other types conversion:

```
public class Program {
    public static void main(String[] args) {
        short shortNumber = 121;

        byte byteNumber = ( byte )shortNumber;
        char character = ( char )shortNumber;

        System.out.println("Byte Number:: "+byteNumber); //121
    }
}
```


Object Oriented Programming with Java

```
System.out.println("Character Value:: "+character);    //y
System.out.println("Short Number:: "+shortNumber);    //121
}
}
```

- byte to other types conversion:

```
public static void main(String[] args) {
    byte byteNumber = 121;

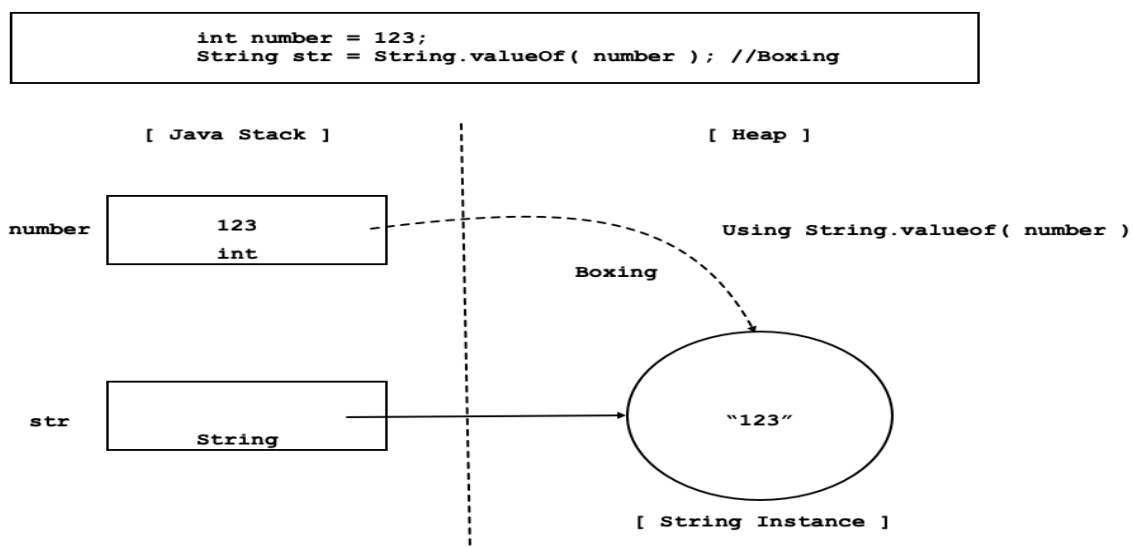
    char character = (char)byteNumber;

    System.out.println("Byte Number:: "+byteNumber);    //121
    System.out.println("Character Value:: "+character); //y
}
```

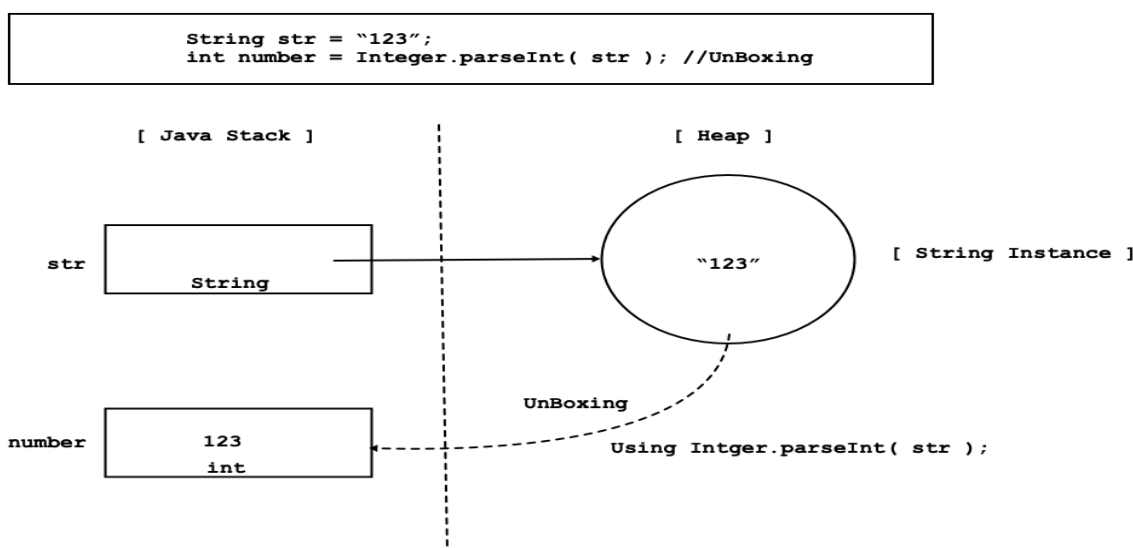
- During narrowing, explicit type casting is required.
- Note:** we can not convert boolean value into any other type and vice versa.

Boxing and Unboxing

- Boxing is the process of converting value of variable of primitive type into non primitive type.



- UnBoxing is the process of converting value of variable of non primitive type into primitive type.



- Note:** We will discuss about boxing/auto-boxing, unboxing/auto-unboxing in detail in generics.

Parsing String Value

- Parsing is the process of converting string value into corresponding numeric value.
- Example 1:

```
public class Program {
    public static void main(String[] args) {
        String str = "true";
        boolean booleanValue = Boolean.parseBoolean(str);
        System.out.println("Boolean Value:: "+booleanValue);
    }
}
```

Object Oriented Programming with Java

- Example 2:

```
public class Program {
    public static void main(String[] args) {
        String str = "A";
        //Character class do not contain parse method.
        char character = str.charAt(0); //Look Here
        System.out.println("Character Value:: "+character);
    }
}
```

- Example 3:

```
public class Program {
    public static void main(String[] args) {
        String str = "2500";
        short shortNumber = Short.parseShort(str);
        System.out.println("Short Number:: "+shortNumber);
    }
}
```

- Example 4:

```
public class Program {
    public static void main(String[] args) {
        String str = "125679";
        int integerNumber = Integer.parseInt(str);
        System.out.println("Integer Number:: "+integerNumber);
    }
}
```

- Example 5:

```
public class Program {
    public static void main(String[] args) {
        String str = "1256.79f";
        float floatNumber = Float.parseFloat(str);
        System.out.println("Float Number:: "+floatNumber);
    }
}
```

- Example 6:

```
public class Program {
    public static void main(String[] args) {
        String str = "975643.79d";
        double doubleNumber = Double.parseDouble(str);
        System.out.println("Double Number:: "+doubleNumber);
    }
}
```

- Example 7:

```
public class Program {
    public static void main(String[] args) {
        String str = "12975643";
        long longNumber = Long.parseLong(str);
        System.out.println("Long Number:: "+longNumber);
    }
}
```

- If String does not contain parsable numeric value then parseXXX() (parseInt(), parseFloat(), parseDouble() etc.)method throws NumberFormatException.

Object Oriented Programming with Java

- To fix NumberFormatException either we should enter proper data or we should do data validation.

```
Program.java X
1 package org.example;
2
3 public class Program {
4     public static void main(String[] args) {
5         String str = "1a2B3c4D";
6         int number = Integer.parseInt(str); //NumberFormatException
7         System.out.println("Number:: "+number);
8     }
9 }
10
```

Problems Javadoc Declaration Console X

<terminated> Program (9) [Java Application] /Users/sandeep/Library/Java/JavaVirtualMachines/corretto-1.8.0_362/Contents/Home/bin/java (22 Feb 2024, 08:35:50 – 08:35:50) [pid: 27397]

Exception in thread "main" java.lang.NumberFormatException: For input string: "1a2B3c4D"
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
at java.lang.Integer.parseInt(Integer.java:580)
at java.lang.Integer.parseInt(Integer.java:615)
at org.example.Program.main(Program.java:6)

- Remember:

- If we want to convert value of any numeric type into another numeric type method call is not required. User C-style type casting.

```
double num1 = 3.14d;
int num2 = ( int ) num1; //OK
```

- If we want to convert value of any non-numeric type into another non-numeric type method call is not required. User C-style type casting. **(We will discuss it in Inheritance)**

```
Number number = new Integer( "123");
Integer integerNumber = (Integer)number;
```

- If we want to convert value of any numeric type into non numeric type or vice versa the method call is required.

```
int number = 123;
String value = String.valueOf( number ); //OK
```

```
String str = "123";
int number = Integer.parseInt( str ); //OK
```

Command line argument

- Example 1:

```
Program.java X
Program.java > Program > main(String[])
1 class Program{
2     public static void main(String[] args){
3         System.out.println( args[ 0 ] );
4     }
5 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

zsh + v [] [] ... ^ X

● sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
● sandeep@Sandeeps-MacBook-Air Day_1 % java Program Hello
Hello
○ sandeep@Sandeeps-MacBook-Air Day_1 %

Object Oriented Programming with Java

- Example 2:

```
J Program.java x
J Program.java > Program > main(String[])
1 class Program{
2     public static void main(String[] args){
3         System.out.println( args[ 0 ] );
4         System.out.println( args[ 1 ] );
5     }
6 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
zsh + v [ ] [ ] ... ^ x
sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
sandeep@Sandeeps-MacBook-Air Day_1 % java Program 10 20
10
20
sandeep@Sandeeps-MacBook-Air Day_1 %
```

- Example 3:

```
J Program.java 2 x
J Program.java > Program
1 class Program{
2     public static void main(String[] args){
3         int num1 = args[ 0 ];
4         int num2 = args[ 1 ];
5         int result = num1 + num2;
6         System.out.println("Result: "+result);
7     }
8 }
```

PROBLEMS **2** OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
zsh + v [ ] [ ] ... ^ x
sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
Program.java:3: error: incompatible types: String cannot be converted to int
    int num1 = args[ 0 ];
                ^
Program.java:4: error: incompatible types: String cannot be converted to int
    int num2 = args[ 1 ];
                ^
2 errors
sandeep@Sandeeps-MacBook-Air Day_1 %
```

- Example 4:

```
J Program.java x
J Program.java > Program
1 class Program{
2     public static void main(String[] args){
3         int num1 = Integer.parseInt( args[ 0 ] );
4         int num2 = Integer.parseInt( args[ 1 ] );
5         int result = num1 + num2;
6         System.out.println("Result: "+result);
7     }
8 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
zsh + v [ ] [ ] ... ^ x
sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
sandeep@Sandeeps-MacBook-Air Day_1 % java Program
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
    at Program.main(Program.java:3)
sandeep@Sandeeps-MacBook-Air Day_1 % java Program 10 20
Result: 30
sandeep@Sandeeps-MacBook-Air Day_1 %
```

- Example 5:

```
J Program.java x
J Program.java > Program > main(String[])
1 class Program{
2     public static void main(String[] args){
3         int num1 = Integer.parseInt( args[ 0 ] );
4         float num2 = Float.parseFloat( args[ 1 ] );
5         double num3 = Double.parseDouble( args[ 1 ] );
6         double result = num1 + num2 + num3;
7         System.out.println("Result: "+result);
8     }
9 }
```

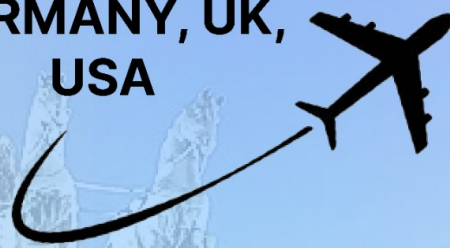
PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
zsh + v [ ] [ ] ... ^ x
sandeep@Sandeeps-MacBook-Air Day_1 % javac Program.java
sandeep@Sandeeps-MacBook-Air Day_1 % java Program 10 20.1f 30.2d
Result: 50.20000038146973
sandeep@Sandeeps-MacBook-Air Day_1 %
```


WHY JUST DREAM?

MAKE IT TRUE WITH US!

STUDY IN
GERMANY, UK,
USA



AWARD
WINNING
INSTITUTE



20,000+
LEARNERS'
COMMUNITY



5+
AFFILIATIONS &
PARTNERSHIPS

- OVERSEAS EDUCATION CONSULTANCY TO UK, USA, GERMANY
- CEFR APPROVED GERMAN LANGUAGE COURSES
- IELTS COACHING (CRACK IELTS IN 30 DAYS)

**EDUWRITE GERMAN LANGUAGE INSTITUTE
& OVERSEAS EDUCATION CONSULTANCY**



9975916717 / 9096799597



www.eduwrite.in



**Pushkaraj Society, Telephone Exchange Road, Behind PICT
College, Dhankawadi, Pune, Maharashtra, India – 411043**



SCAN FOR DETAILS

