

# Object Oriented Programming using Java

**Comprehensive overview of Java**



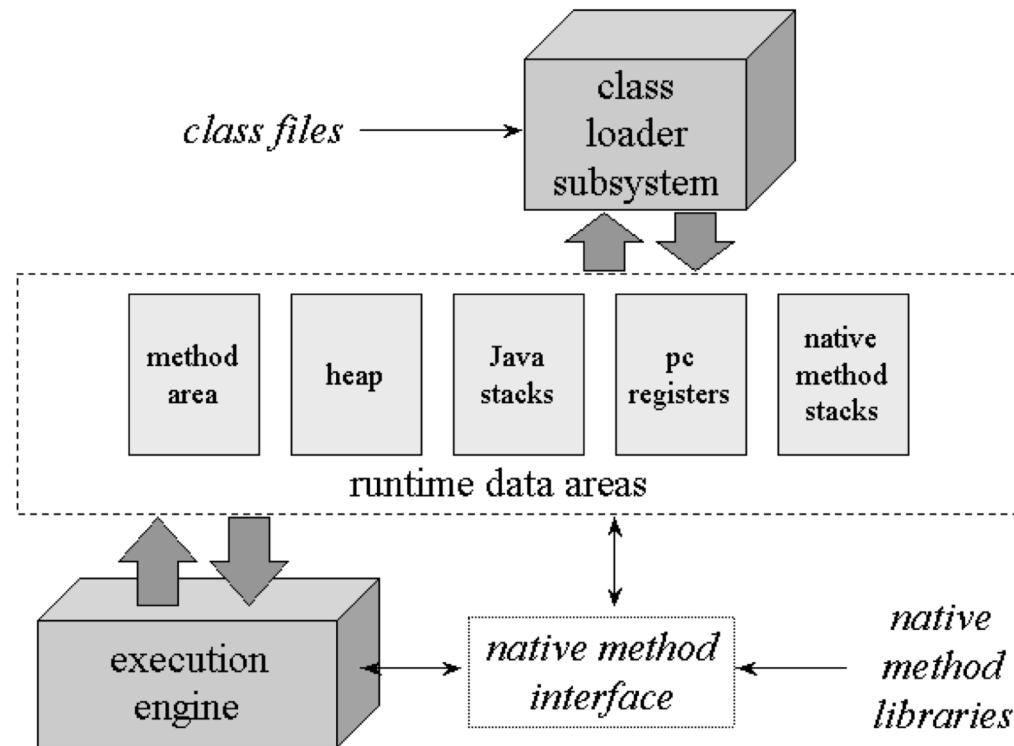
[sandeepkulange@gmail.com](mailto:sandeepkulange@gmail.com) | 9527325202

# Session Overview

- Components of JVM
- Java Buzzwords
- Java Modifier
- Access Modifier
- Java Virtual Machine Threads
- Entry Point Method
- Meaning of System.out.println
- Data Types
- Classification of Data Types
- Primitive Data Types
- Wrapper Class Hierarchy
- Overview of String
- Memory representation
- Widening Conversion
- Narrowing Conversion
- Boxing
- UnBoxing
- Command line Arguments

# Components of JVM

- **Class loader subsystem**
  1. Bootstrap class loader
  2. Extension class loader
  3. System class loader
  4. Custom class loader
- **Runtime data areas**
  1. Method area
  2. Heap
  3. Java Stacks
  4. PC Register
  5. Native method stacks
- **Execution engine**
  1. Interpreter
  2. Just In Time Compiler
  3. Garbage Collector



# Java Buzzwords

- *Java Buzzwords* are Java language marketing words.

1. Simple
2. Object Oriented
3. Architecture Neutral
4. Portable
5. Robust
6. Dynamic
7. Multithreaded
8. Secure
9. High Performance
10. Distributed

# Java Modifier

- A keyword which is used to change the behavior of variable, field, method, class etc.
- There are 12 modifiers in Java:

1. private	:	Access Modifier
2. protected	:	Access Modifier
3. public	:	Access Modifier
4. static	:	Non Access Modifier
5. final	:	Non Access Modifier
6. abstract	:	Non Access Modifier
7. interface	:	Non Access Modifier
8. transient	:	Non Access Modifier
9. synchronized	:	Non Access Modifier
10. volatile	:	Non Access Modifier
11. strictfp	:	Non Access Modifier
12. native	:	Non Access Modifier

# Access Modifier

- Modifier which is used to control visibility of the members of the class/enum/interface.
- 4 Access modifiers in Java
  1. private
  2. package level private( also called as default )
  3. protected
  4. public

Access Modifier	Same Package			Different Package	
	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

```
sandeep@Sandeeps-MacBook-Air Day_1.1 % cat Program.java
class Program{
    public static void main( String[] args )throws Exception{
        System.out.println("Press any key to continue...");
        System.in.read( );
    }
}
sandeep@Sandeeps-MacBook-Air Day_1.1 %
sandeep@Sandeeps-MacBook-Air Day_1.1 % jcmd
44201 sun.tools.jcmd.JCmd
44173 Program
sandeep@Sandeeps-MacBook-Air Day_1.1 %
sandeep@Sandeeps-MacBook-Air Day_1.1 % jstack 44173
```

# Java Virtual Machine Threads

## 1. [Attach Listener](#)

- A daemon thread responsible for handling the attachment of debuggers.

## 2. [Service Thread](#)

- A daemon thread that handles JVM service tasks

## 3. [C2 CompilerThread0](#)

## 4. [C2 CompilerThread1](#)

## 5. [C1 CompilerThread2](#)

- These are compiler threads responsible for compiling Java bytecode to native code.

## 6. [Signal Dispatcher](#)

- A daemon thread that handles OS signals.

## 7. [Main](#)

- main application thread.

## 8. [VM Thread](#)

- A JVM thread used for internal JVM tasks.

## 9. [GC task thread#0 \(ParallelGC\)](#)

## 10. [GC task thread#1 \(ParallelGC\)](#)

## 11. [GC task thread#2 \(ParallelGC\)](#)

## 12. [GC task thread#3 \(ParallelGC\)](#)

- Threads dedicated to garbage collection

# Entry Point Method

- “main” method is considered as entry point method in java.
- Legal main method signatures
  1. `public static void main( String[] args )`
  2. `public static void main( String args[ ] )`
  3. `public static void main( String... args )`
- Java compiler do not call main method. With the help of main thread JVM invoke main method in Java.
- If we try to execute class which do not contain main method then we get below error:
  - **Error: Main method not found in class Program, please define the main method as:**  
`public static void main(String[] args)`
- We can overload main method in Java.

# Meaning of System.out.println

```
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();
    }
}
```

```
package java.io;

public class PrintStream {

    public void print(String s) {
        // Method to print a string without a newline
    }

    // Other overloaded print(...) methods

    public void println(String s) {
        // Method to print a string followed by a newline
    }

    // Other overloaded println(...) methods

    public PrintStream printf(String format, Object... args) {
        // Method to print formatted strings
        return this;
    }

    // Other overloaded printf() methods
}
```

# Data Types

- Java is statically as well as strongly typed language.
- Data Type of any variable describes following properties:
  - **Memory Allocation:** How much memory is required to store the data?
  - **Type of Data:** What kind of data is allowed to store inside variable?
  - **Operation:** Which operations can be performed on the data stored in memory?
  - **Range of data:** Range of values that can be stored in a variable?
- Classification of data types:
  1. **Primitive Data Types**(also called as value types)
  2. **Non Primitive Data Types**(also called as reference types)

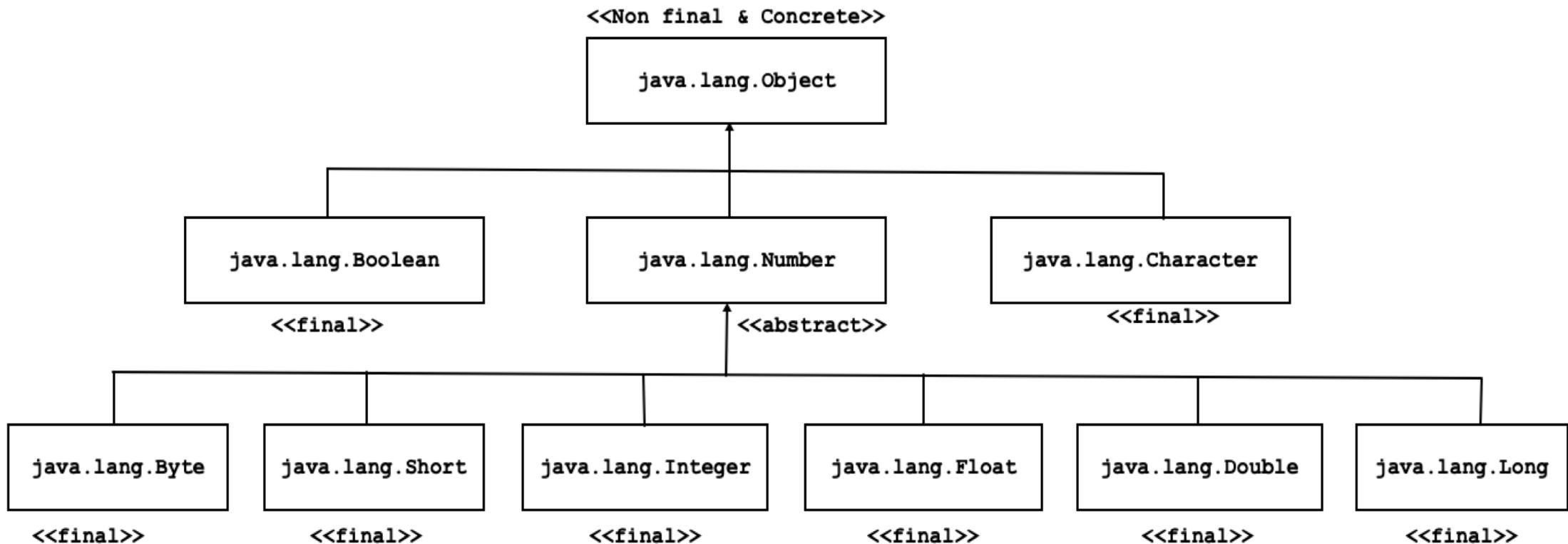
# Classification of Data Types

- Primitive Data Types
  - 1. boolean
  - 2. byte
  - 3. char
  - 4. short
  - 5. int
  - 6. float
  - 7. double
  - 8. Long
- Variable of above type can contain only value. Hence such type is also called as value type.
- Reference:  
<https://docs.oracle.com/javase%2Ftutorial%2Fjava/nutsandbolts/datatypes.html>
- Non Primitive Data Types
  - 1. Interface
  - 2. Class
  - 3. Array
  - 4. Enum
- Variable of above type can contain only object reference/reference. Hence such type is also called as reference type.
- Reference:  
<https://docs.oracle.com/javase/tutorial/reflect/class/index.html>

# 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

# Wrapper Class Hierarchy



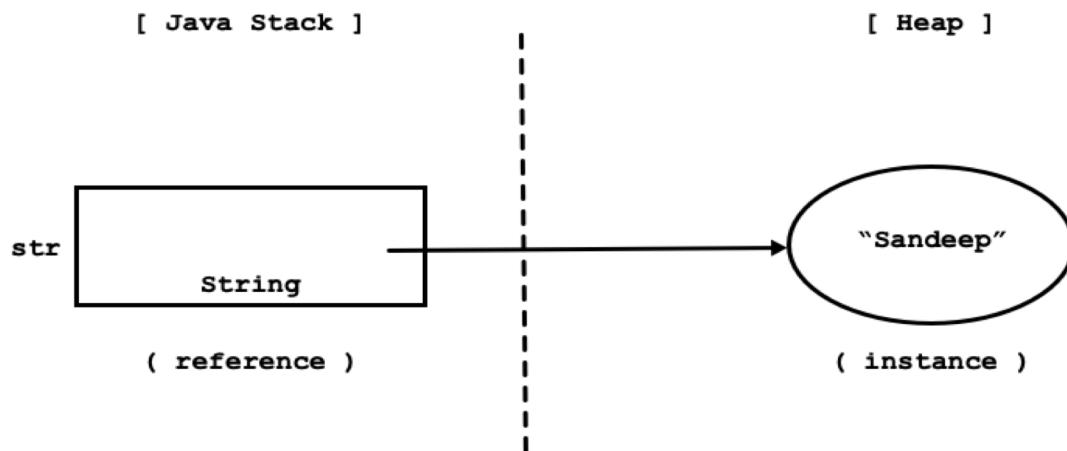
# Overview of String

- String is not a primitive / built-in type in Java.
- String is a final class declared in java.lang package.
- Since String is a class, it is considered as non primitive type/reference type.
- We can create Instance of String using new operator as well as without new operator.
- Example 1:
  - `String str = new String("Sandeep"); //OK`
  - `str` is called as object reference / simply reference.
  - `new String("Sandeep")` is called as instance.
- Example 2:
  - `String str = "Sandeep"; //OK`
  - `str` is called as object reference / simply reference.
  - `"Sandeep"` is called as String constant/literal.

# Memory Representation

- Example 1:

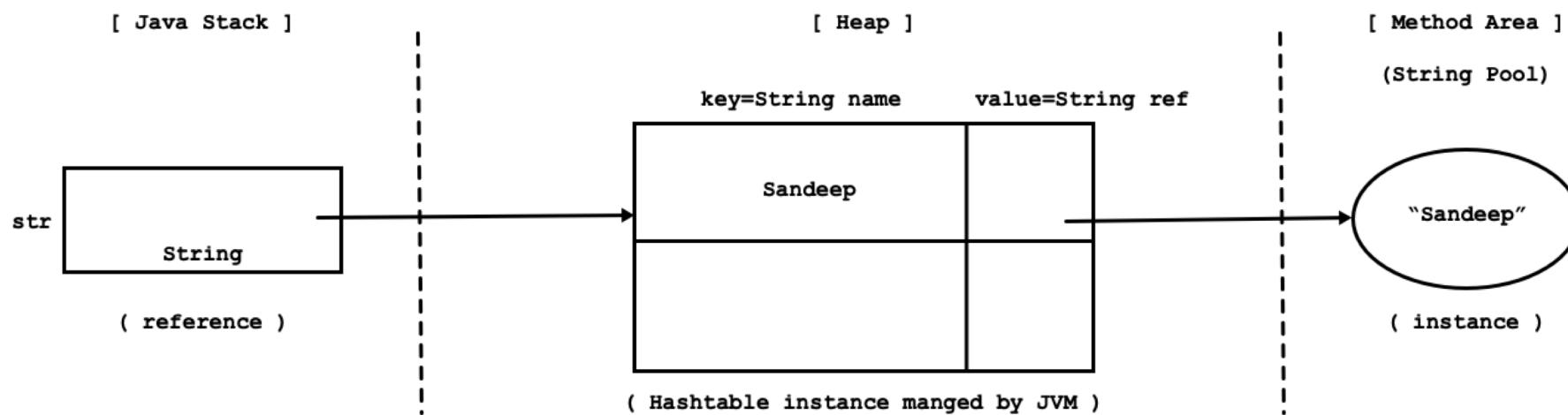
- `String str = new String("Sandeep"); //OK`
  - `str` is called as object reference / simply reference.
  - `new String("Sandeep")` is called as instance.



# Memory Representation

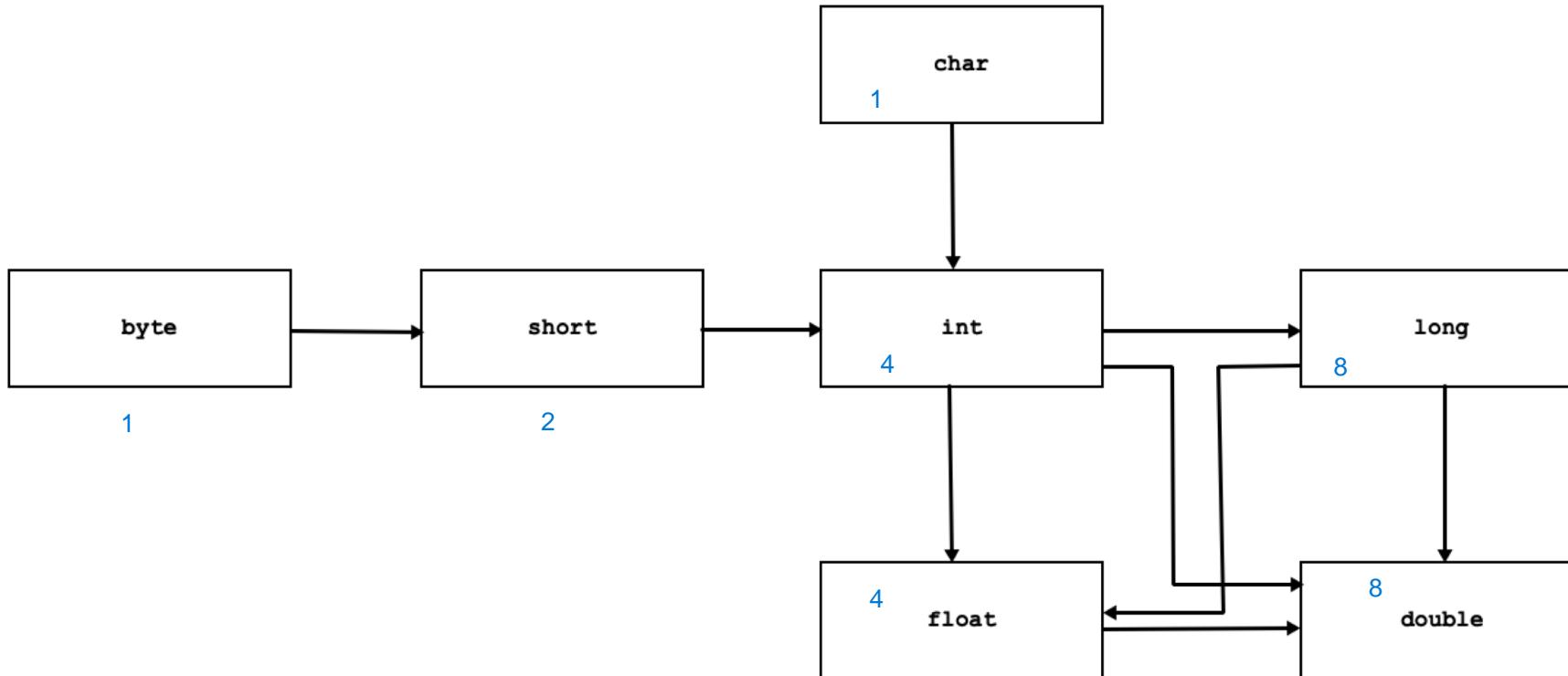
- Example 2:

- String str = "Sandeep"; //OK
  - str is called as object reference / simply reference.
  - "Sandeep" is called as String constant/literal.



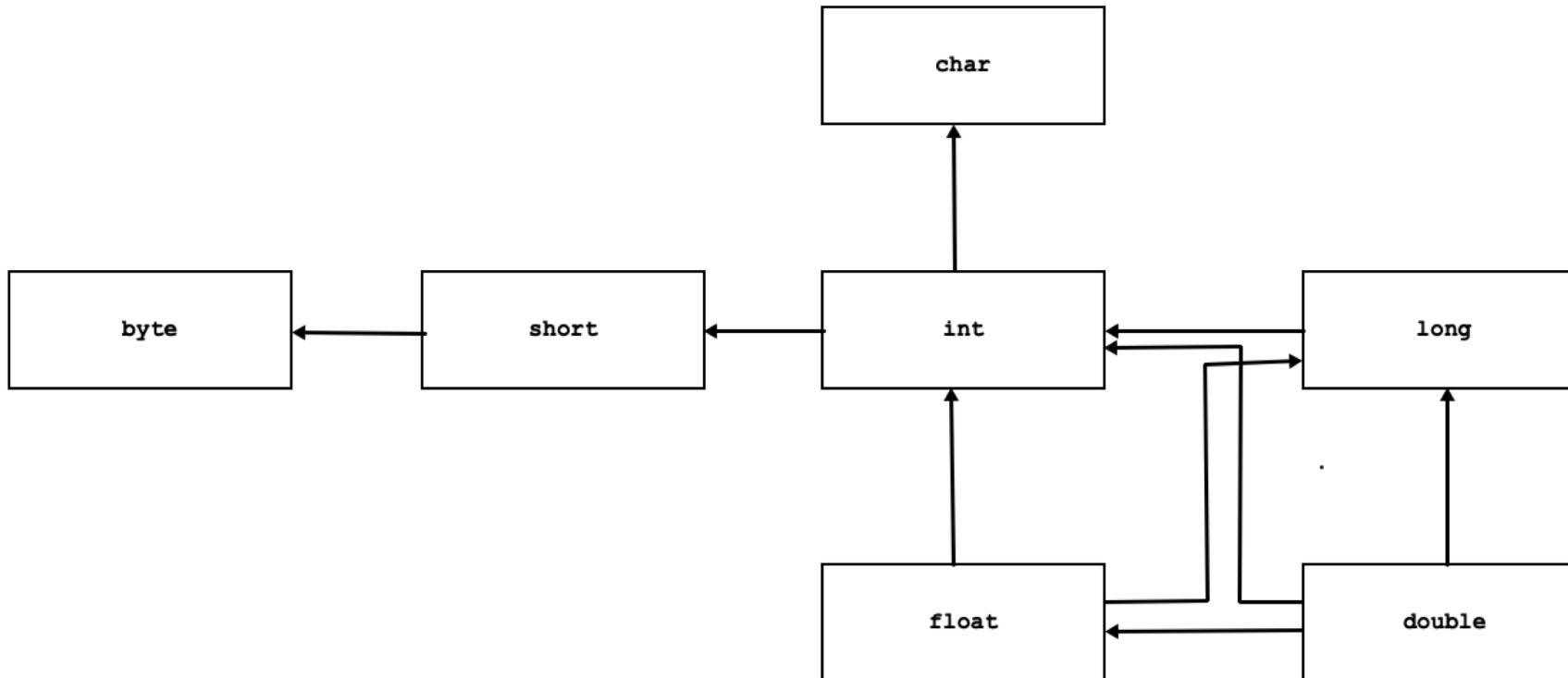
# Widening Conversion

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



# Narrowing Conversion

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



# Boxing

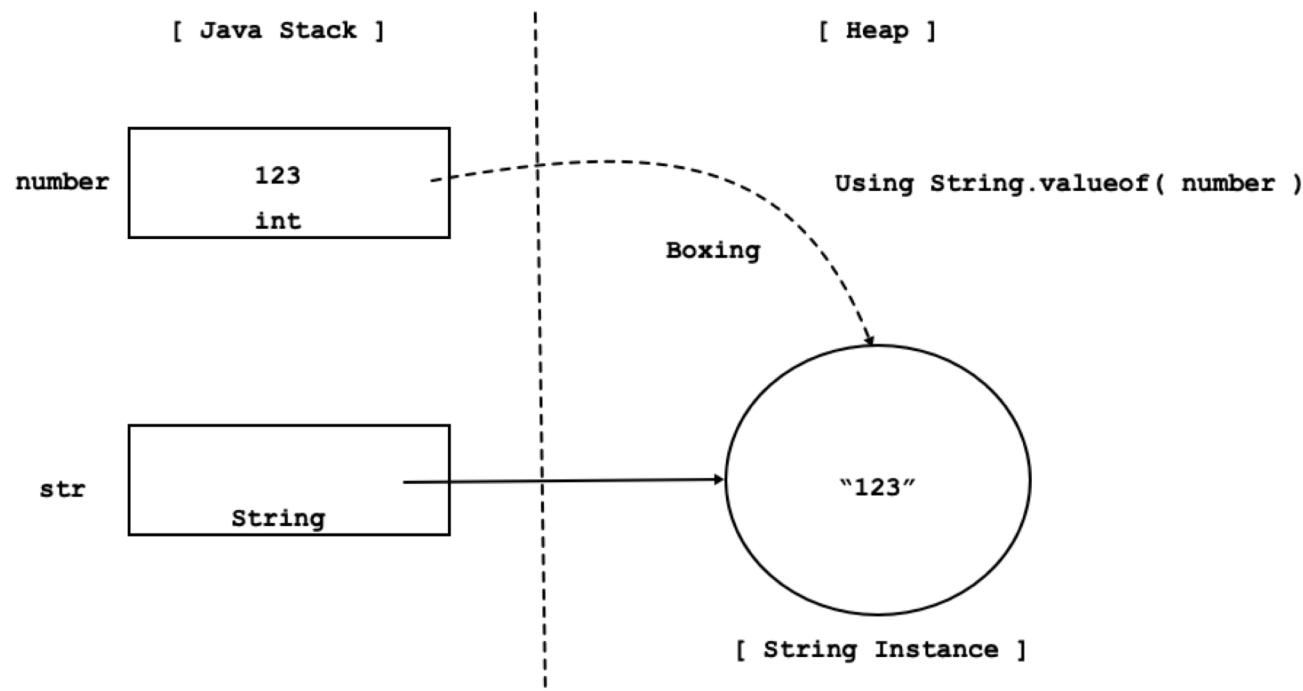
- Using `toString()` method of Wrapper class or `String.valueOf()` method, we can convert value of any primitive type into String.

1. `String s1 = Boolean.toString( true );` **or** `String s1 = String.valueOf( true );`
2. `String s2 = Character.toString( 'A' );` **or** `String s2 = String.valueOf( 'A' );`
3. `String s3 = Integer.toString( 123 );` **or** `String s3 = String.valueOf( 123 );`
4. `String s4 = Float.toString( 123.45f );` **or** `String s4 = String.valueOf( 123.45f );`
5. `String s5 = Double.toString( 123.45d );` **or** `String s5 = String.valueOf( 123.45d );`

# Boxing

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

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



# Unboxing

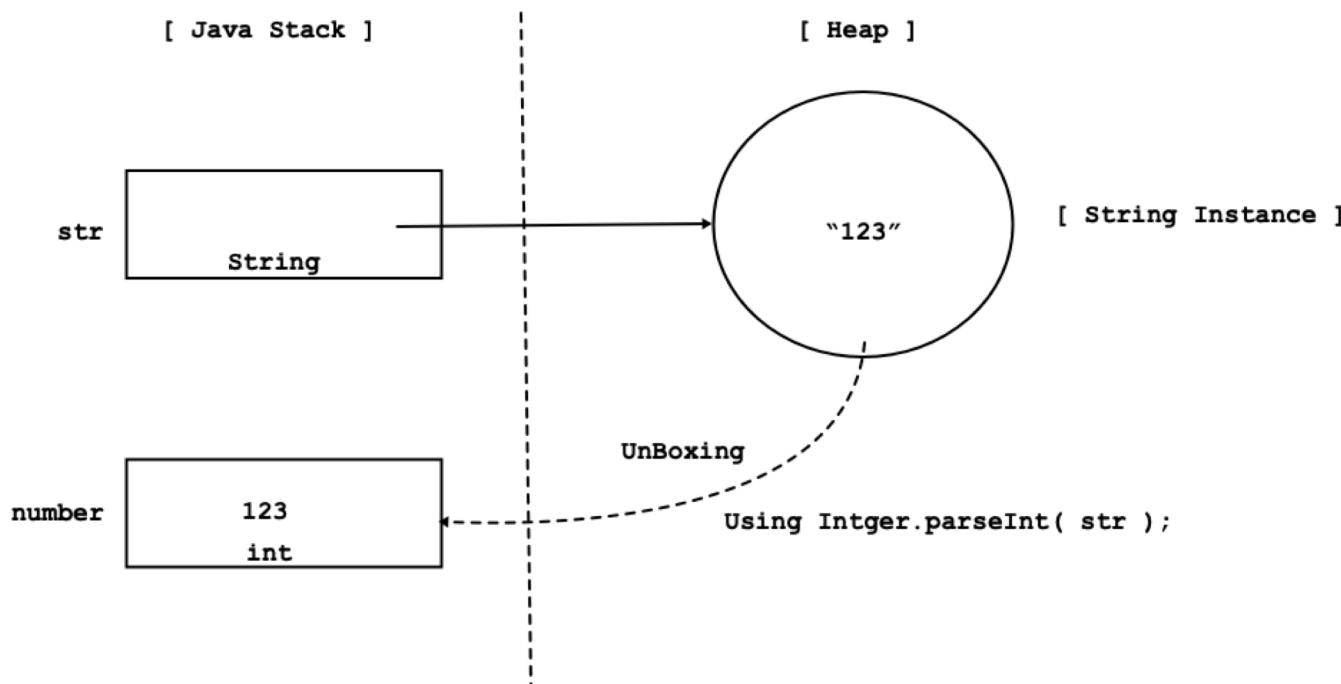
- Using `parseXXX()` method of Wrapper class, we can convert state of String into primitive value.

```
1. boolean b = Boolean.parseBoolean("true");  
  
2. int i = Integer.parseInt( "123" );  
  
3. float f = Float.parseFloat( "123.45f" );  
  
4. double d = Double.parseDouble( "123.45d" );  
  
5. int number = Integer.parseInt( "1A2b3C" ); //NumberFormatException
```

# Unboxing

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

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



# Command line Arguments

A screenshot of a Java code editor interface. The code editor window shows a file named "Program.java" with the following content:

```
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     }
5 }
```

The code editor has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is currently selected, displaying the following terminal session:

- sandeep@Sandeeps-MacBook-Air Day\_1 % javac Program.java
- sandeep@Sandeeps-MacBook-Air Day\_1 % java Program Hello
- sandeep@Sandeeps-MacBook-Air Day\_1 % █

# Command line Arguments

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** J Program.java X
- Code Area:** Shows the following Java code:

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

A yellow warning icon is positioned next to the second line of code.
- Toolbars:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (underlined), PORTS
- Terminal:** Shows the following command-line session:

```
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 %
```
- Right Panel:** Includes icons for zsh, terminal, file operations, and others.

# Command line Arguments

J Program.java 2 ×

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 + ~ || ⌂ ... ^ ×

```
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

# Command line Arguments

A screenshot of a Java code editor showing a file named Program.java. The code defines a class Program with a main method that adds two command-line arguments and prints the result. The code editor interface includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS, and a terminal window below showing the execution of the 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  }
```

```
● sandeep@Sandeeeps-MacBook-Air Day_1 % javac Program.java
✖ sandeep@Sandeeeps-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@Sandeeeps-MacBook-Air Day_1 % java Program 10 20
Result::30
○ sandeep@Sandeeeps-MacBook-Air Day_1 %
```

# Command line Arguments

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** J Program.java X
- Code Area:** The code is a Java program named `Program` that takes three command-line arguments and prints their sum.

```
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[ 2 ] );
6         double result = num1 + num2 + num3;
7         System.out.println("Result::"+result);
8     }
9 }
```
- Terminal Tab:** The terminal tab shows the execution of the program:
  - Line 1: sandeep@Sandeeps-MacBook-Air Day\_1 % javac Program.java
  - Line 2: sandeep@Sandeeps-MacBook-Air Day\_1 % java Program 10 20.1f 30.2d
  - Line 3: Result::50.20000038146973
  - Line 4: sandeep@Sandeeps-MacBook-Air Day\_1 % █

