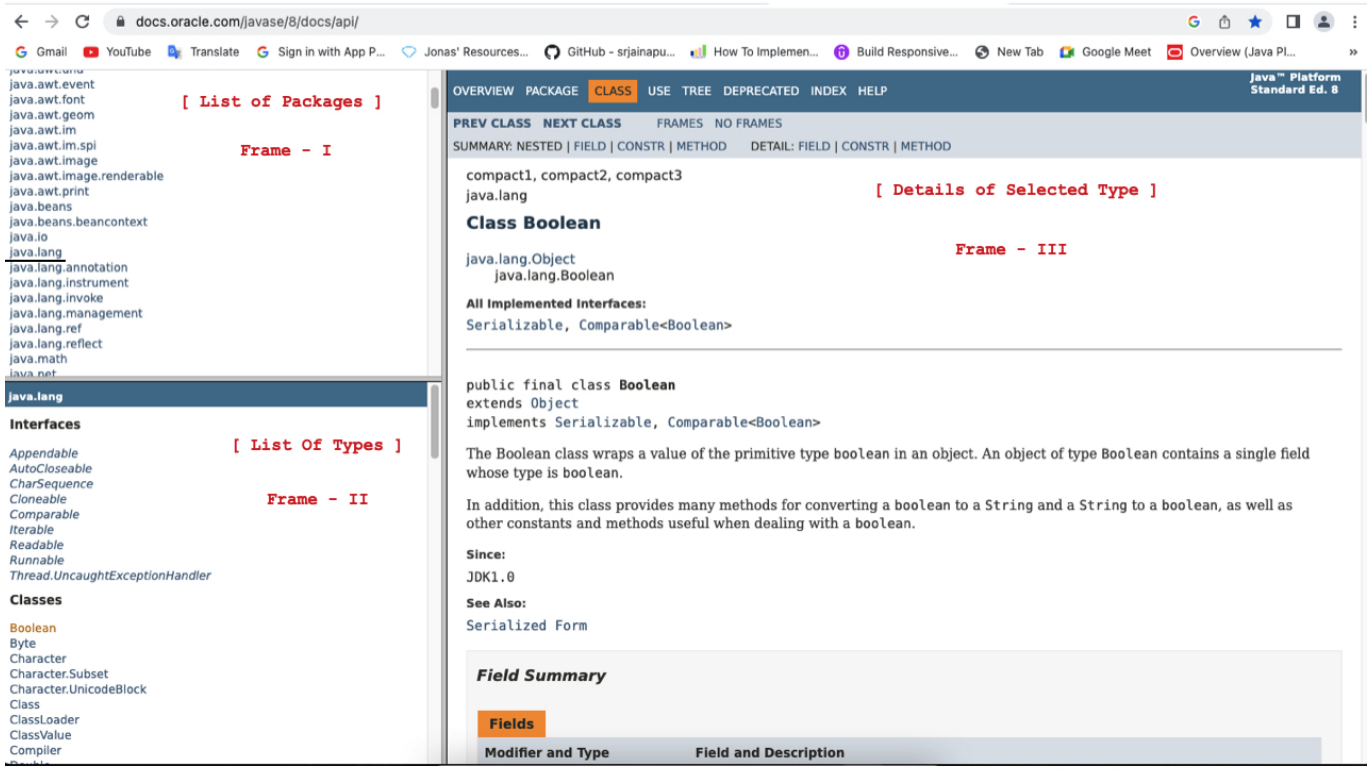
Day 2

Java Documentation



Contents of package

- Sub package
- Interface
    - Nested Type
    - Constant Fields
    - Abstract Methods
    - Default Methods
    - Static Interface Methods
- Class
    - Nested Types(Interface / Class / Enum)
    - Fields
    - Constructor
    - Method
- Enum
- Exception

- Error
- Annotation Type

Modifiers in Java

- PUBLIC
- PRIVATE
- PROTECTED
- STATIC
- FINAL
- SYNCHRONIZED
- VOLATILE
- TRANSIENT
- NATIVE
- INTERFACE
- ABSTRACT
- STRICT
- Reference: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Modifier.html

Access Modifiers

- Modifiers which are used to control visibility of members of the class is called access modifier.
- Access modifiers
  - private
  - Package level private(also called as default)
  - protected
  - public

More about main method

- In java, main method is considered as entry point method.
- With the help of main thread, JVM invoke main method.
- Syntax:
  - public static void main(String args[])
  - public static void main(String[] args)
  - public static void main(String... args)
- We can define main method inside class as well as interface.

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

```
interface Program{
  public static void main(String[] args ) {
```

```
    System.out.println("Hello World!!!");
  }
}
```

- Reference: https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Java Comments

- To maintain documentation of source code we should use comments:
- Types of comments:
  - Implementation Comments
    - Block comments / multiline comments

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

    - Single line comments

```
class Program{
  public static void main(String[] args ) {
    //System.out.println("Hello World!!!");
    System.out.println("Good Morning");
  }
}
```

  - Documentation comment / doc comments

```
/**
  Author is Sandeep Kulange
*/
class Program{
  /**
  Entry point method of class Program
  */
  public static void main(String[] args ) {
    System.out.println("Hello World!!!");
  }
}
```

- Reference: https://www.oracle.com/java/technologies/javase/codeconventions-comments.html

Java OOPS concepts

- Consider following code:

```
class Test{
  //Fields
  private int num1;          //Instance variable
  private static int num2;  //Class level variable
  //Methods
  public void setNum1( int num1 ){  //Instance method
    this.num1 = num1;
  }
  public static void setNum2( int num2 ){ //Class level method
    this.num2 = num2;
  }
}
```

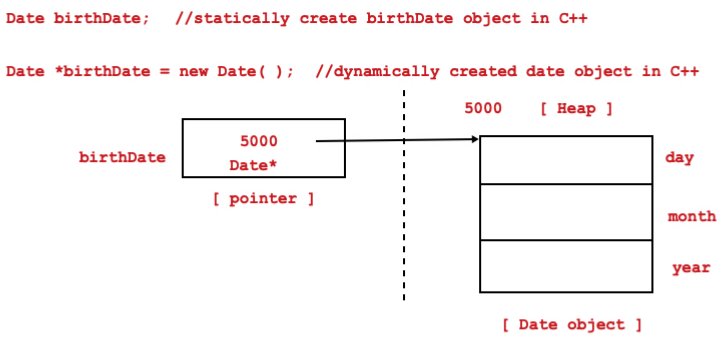
- In Java, to access static members (class level members) of the class, we must use class name and dot operator.

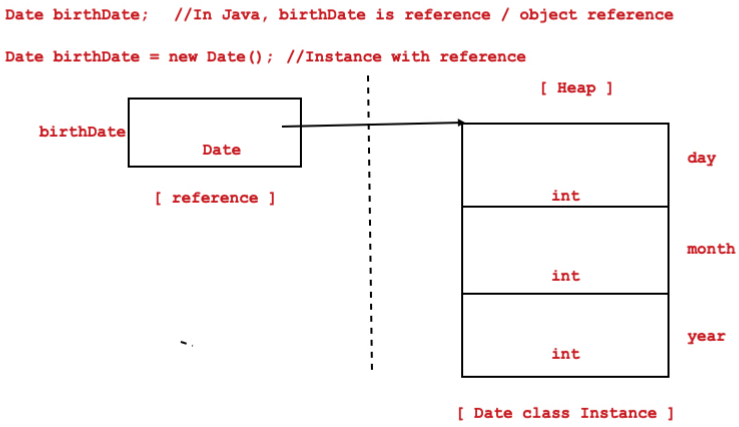
```
public static void main(String[] args) {
  Test.setNum2( 123 );
}
```

- To access non static members(instance members) of the class, we must use instance of the class.

```
  Test t; //Object reference / reference
  new Test; //Not OK
  new Test(); //OK: Instance
  Test t = new Test( ); //Instantiation
```

```
public static void main(String[] args) {
  //Test.setNum1( 100 );  //Not OK
  Test t = new Test( );
  t.setNum1( 100 ); //OK
}
```





- A class from which we can not create instance(In C/C++ it is called as object) is called abstract class.
- In simple words, we can not instantiate abstract class.
- Example:
  - java.lang.Number
  - java.lang.Enum
- A class from which we can create instance is called concrete class.
- In simple words, we can instantiate concrete class.
- Example:
  - java.lang.Runtime
  - java.lang.Thread
- A method of class which is having body is called as concrete method.
- main method is concrete method.
- Example:

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

- A method of class which do not have body is called as abstract method.

- Example:

```
abstract class Shape{
  public abstract void calculateArea( );
}
```

- A class from which we can not create child / sub class is called as final class.
- We can instantiate final class.
- Example:
  - java.lang.System
  - java.lang.String
- If method in parent /super class is final then we can not redefine / override that method in child / sub class.

What is System.out.println?

```
package java.lang;
public final class System {
    //Fields
    public final static InputStream in = null;

    public final static PrintStream out = null;

    public final static PrintStream err = null;
```

- System is a final class declared in java.lang package.
- out is a reference of java.io.PrintStream class. It is declared as public static final field inside System class.

```
  //How will you access in
  System.in //It reprsents keyboard
  //How will you access out
  System.out  //It represents Monitor
  //How will you access err
  System.err  //It represents Monitor for errors
```

- PrintStream is a class declared in java.io package.

```
package java.io;
public class PrintStream extends FilterOutputStream implements
Appendable, Closeable{

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);

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);

public PrintStream printf(String, Object...);
public PrintStream printf(Locale, String, Object...);
}
```

- println is a non static method of java.io.PrintStream class.

print, println and printf

- System.out.print method print output on console but it keeps cursor on same line.

```
class Program{
  public static void main1(String[] args ) {
    System.out.print("Hello ");
    System.out.print(" World");
    System.out.print("!!");
  }
}
```

- System.out.println method print output on console but it moves cursor to the next line.

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

- To print formatted output on console / terminal, we should use printf method.

```
    public static void main(String[] args) {
      String name1 = "Sandeep Kulange";
      int empid1 = 10003778;
      float salary1 = 45000.50f;

      String name2 = "Mahesh Koli";
      int empid2 = 3779;
      float salary2 = 125000.50f;


      System.out.printf("%-20s%-10d%-10.2f\n", name1, empid1,
salary1);
      System.out.printf("%-20s%-10d%-10.2f\n", name2, empid2,
salary2);
    }
```

Data Types

- Data type of any variable describe following things:
    - (Memory) How much memory is required to store the data.
    - (Nature) Which type of data is allowed to store inside allocated memory.
    - (Range) How much data is allowed to store inside allocated memory.
    - (Operation) Which operations are allowed to perform / execute on the data stored inside memory.
- Types of data type
    - Primitive data type
        - Variable of primitive data type contains value. Hence primitive type is also called as value type.
        - There are 8 primitive / value types in Java:
            - boolean
            - byte
            - char
            - short
            - int
            - float
            - double
            - long
    - Non Primitive data type
        - Variable of non primitive data type contains reference of the instance. Hence non primitive type is also called as reference type.
        - There are 4 non primitive / reference types in Java
            - Interface
            - Class
            - Enum
            - Array

- If we declare variable inside method then it is called as method local variable.
    - In Java, we can not declare method local variable static.

```
class Program {
  public static void main(String[] args) {
    int number; //Non Static Method Local Variable  //OK
  }
}
```

```
class Program {
  public static void main(String[] args) {
    int number; //Non Static Method Local Variable
    System.out.println( number ); //error: variable number might not
have been initialized
  }
}
```

- If we want to use any method local variable(primitive / non primive) then we must store some value inside it.

```
class Program {
  public static void main(String[] args) {
    int number = 10; //Initialization
    System.out.println( number ); //OK
  }
}
```

```
class Program {
  public static void main(String[] args) {
    int number;
    number = 10;  //Assignment
    System.out.println( number ); //OK
  }
}
```

Sr.No.	Primitive Type	Size	Default for field	Wrapper Class
1	boolean	Not Specified	FALSE	java.lang.Boolean
2	byte	1 byte	0	java.lang.Byte
3	char	2 Bytes	\u0000	java.lang.Character
4	short	2 Bytes	0	jav.lang.Short
5	int	4 Bytes	0	java.lang.Integer
6	float	4 Bytes	0.0f	java.lang.Float
7	double	8 Bytes	0.0d	java.lang.Double
8	long	8 Bytes	0L	java.lang.Long

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

Wrapper classes

- In Java, primitive types are not classes. But Sun/Oracle developers has written classes for it. Such classes are called as Wrapper classes.
- All the Wrapper classes are final and declared in java.lang package.
- Class Hierarchy
    - java.lang.Object
        - java.lang.Boolean
        - java.lang.Character
        - java.lang.Number
            - java.lang.Byte
            - java.lang.Short
            - java.lang.Integer
            - java.lang.Float
            - java.lang.Double
            - java.lang.Long

Initialization

- Process of storing value inside variable during its declaration is called initialization.

```
int number = 10;  //Initialization
```

```
int num1 = 10;    //Initialization
int num2 = num1;  //Initialization
```


```
int num1 = 10;    //Initialization
int num2 = num1;  //Initialization
```

```
//int num2 = 20;  //error: variable num2 is already defined in method
main(String[])
```

- We can initialize variable only once.
- Process of storing value inside variable after its declaration is called as assignment.

```
  public static void main(String[] args) {
  int num1 = 10;    //Initialization
  int num2 = num1;  //Initialization
  num2 = 20;  //Assignment  : OK
  num2 = 30;  //Assignment  : OK
}
```

- We can do assignment multiple times.

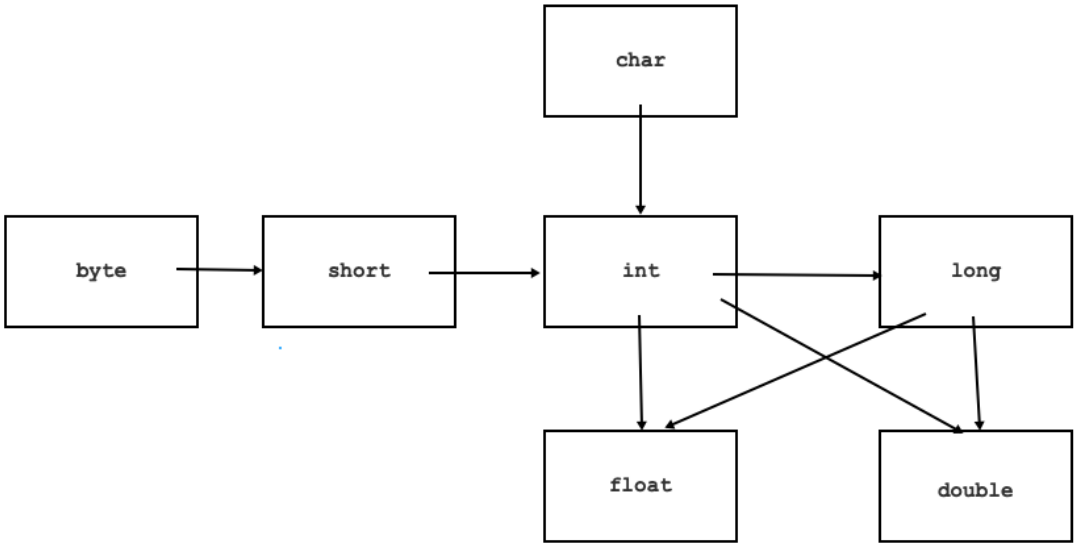
Narrowing & Widening

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

```
  public static void main(String[] args) {
    int num1 = 10;
    double num2 = ( double )num1; //Widening
    System.out.println("Num2  : "+num2);  //OK: Num2  : 10.0
  }
```

- In case of widening conversion, explicit typecasting is optional.

```
  public static void main(String[] args) {
    int num1 = 10;
    //double num2 = ( double )num1; //Widening
    double num2 = num1; //Widening
    System.out.println("Num2  : "+num2);  //OK: Num2  : 10.0
  }
```

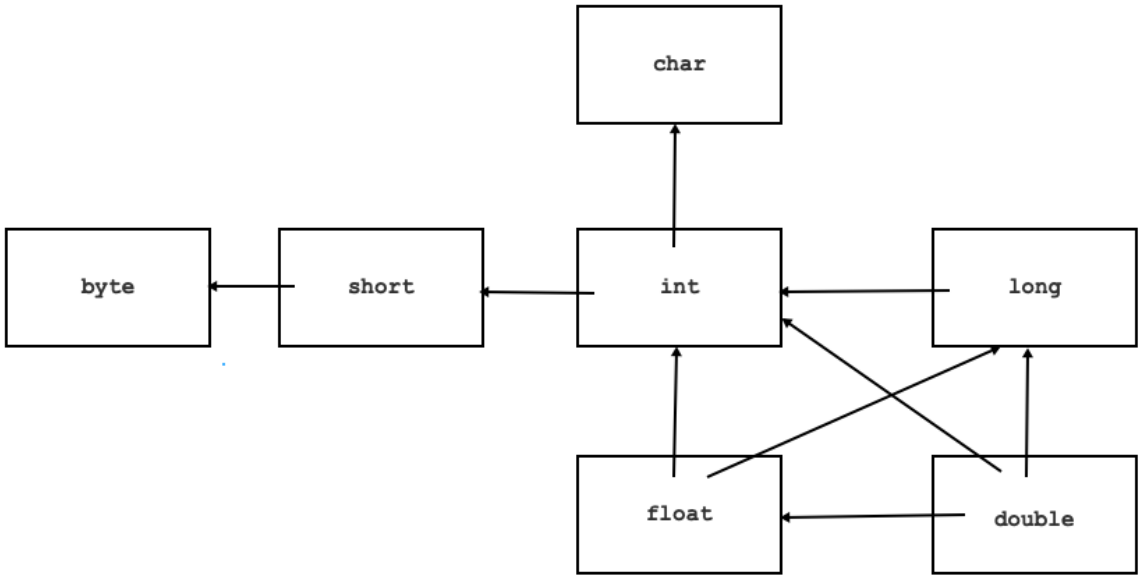


- Process of converting value of variable of wider type into narrower type is called as narrowing.

```
public static void main(String[] args) {
  double num1 = 10.5d;
  int num2 = (int)num1; //Narrowing
  System.out.println("Num2  : "+num2);
}
```

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

```
  public static void main(String[] args) {
    double num1 = 10.5d;
    int num2 = (int)num1; //Narrowing
    //int num2 = num1; //error: incompatible types: possible lossy
conversion from double to int
    System.out.println("Num2  : "+num2);
  }
```

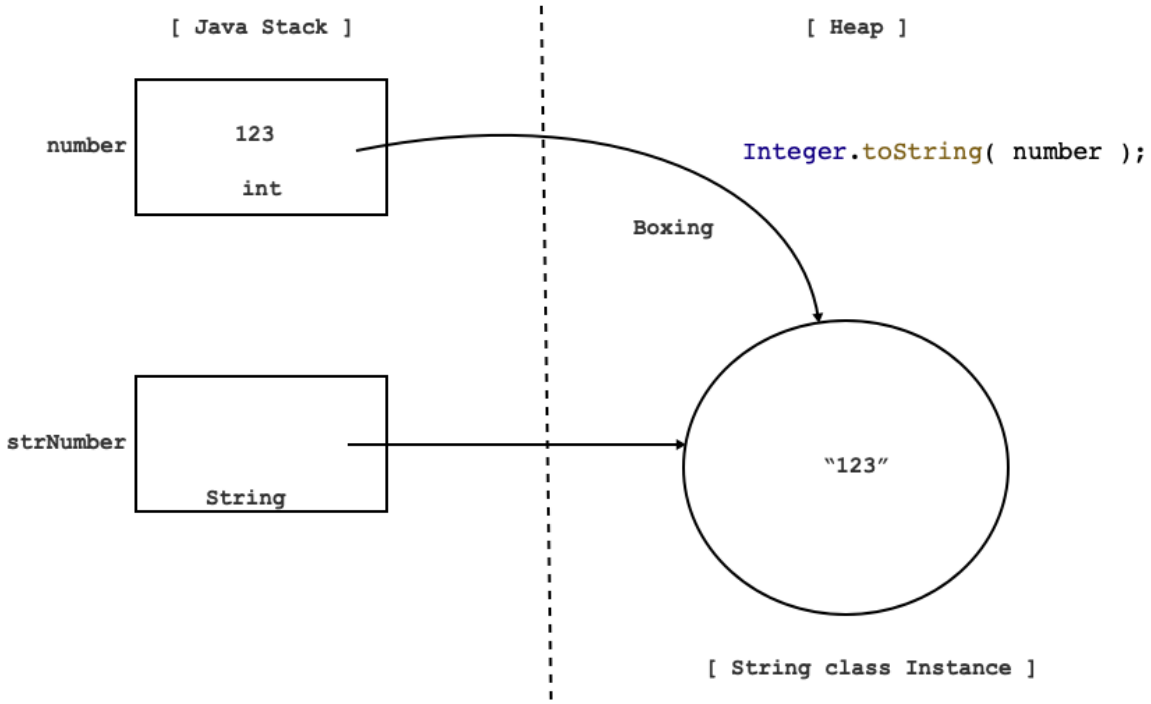


Boxing & Unboxing

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

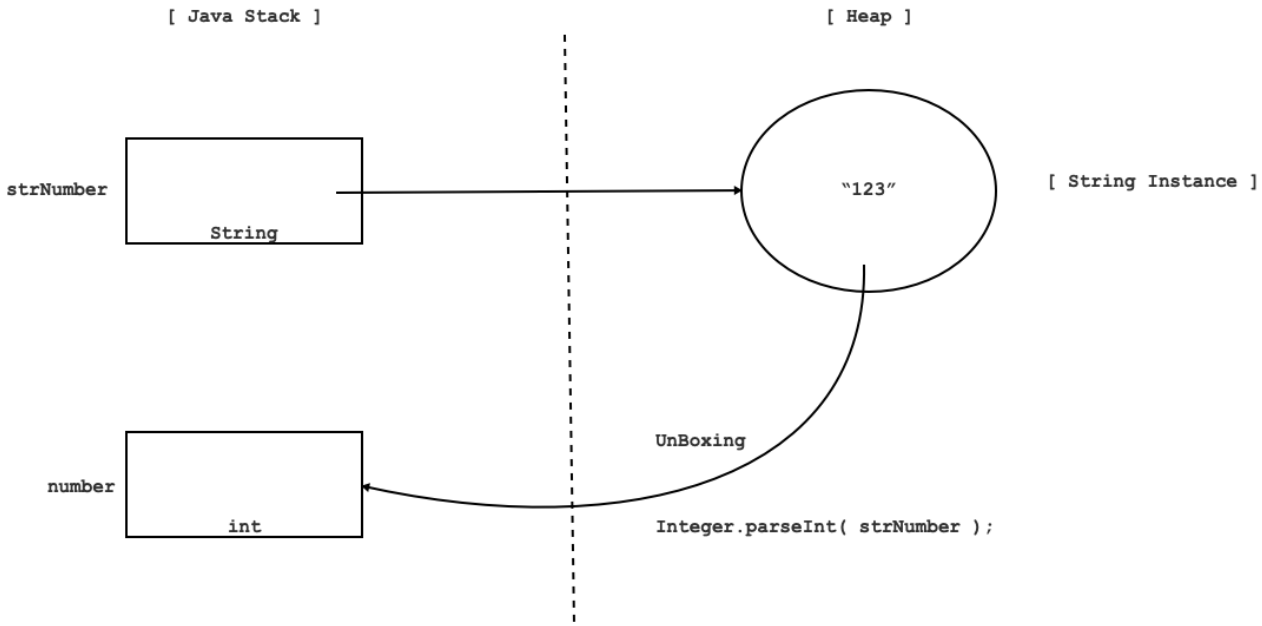
```
  public static void main(String[] args) {
    int number = 123;
    //String str = ( String)number; //error: incompatible types: int
cannot be converted to String
    String strNumber = Integer.toString( number );  //Boxing
    System.out.println("Number  : "+strNumber);
  }
```

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



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

```
  public static void main(String[] args) {
  String strNumber = new String( "123" );
  //int number = ( int ) strNumber; //error: incompatible types:
String cannot be converted to int
  int number = Integer.parseInt(strNumber); //UnBoxing
  System.out.println("Number  : "+number);
}
```



NumberFormatException

```
  public static void main(String[] args) {
    String strNumber = new String( "1A2B3C" );
    int number = Integer.parseInt(strNumber);
//java.lang.NumberFormatException
    System.out.println("Number  : "+number);
  }
```

- If the string does not contain a parsable numeric value then parseXXX() method throws NumberFormatException. Reference: https://docs.oracle.com/javase/8/docs/api/java/lang/NumberFormatException.html

Commandline arguments

- Consider code in C language

```
int sum( int num1, int num2 ){  //num1, num2  <=  Function parameters
  int result = num1 + num2;
  return result;
}
int main( void ){
  int result = sum( 10, 20 ); //10, 20 <= Function arguments
  return 0;
}
```

- How to pass argument from command line?

```
sandeep@Sandeeps-MacBook-Air Day_2.6 % java Program Sandeep
```

```
class Program {
  //args is a method parameter
  public static void main(String[] args) {
    System.out.println("Hello,"+args[ 0 ]);
  }
}
```

```
sandeep@Sandeeps-MacBook-Air Day_2.6 % java Program "Sandeep Kulange"
```

```
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);
  }
  public static void main1(String[] args) {
    int num1 = Integer.parseInt(args[ 0 ]);
    int num2 = Integer.parseInt(args[ 1 ]) ;
    int result = num1 + num2;
    System.out.println("Result  : "+result);
  }
}
```