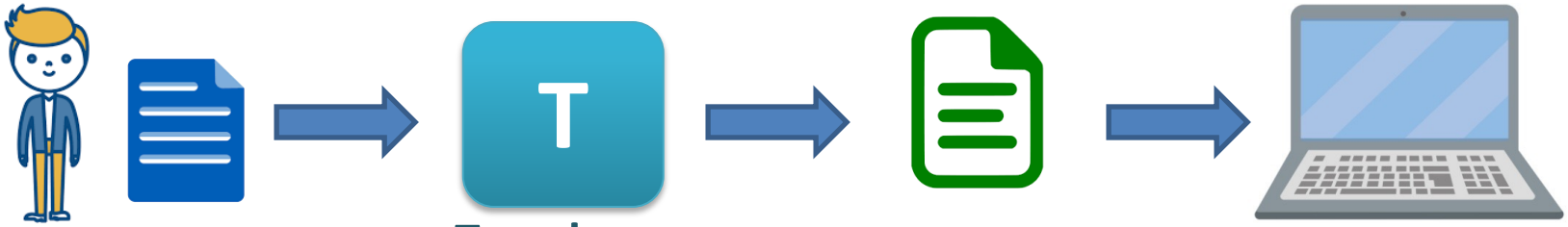
Programming Language (PL)

- **PL is an artificial language used to communicate with computers(Machines).**
- **PL is used to build software.**



Java Introduction & History

Java Programming Language

Created By: **James Gosling (Dr. Java)**

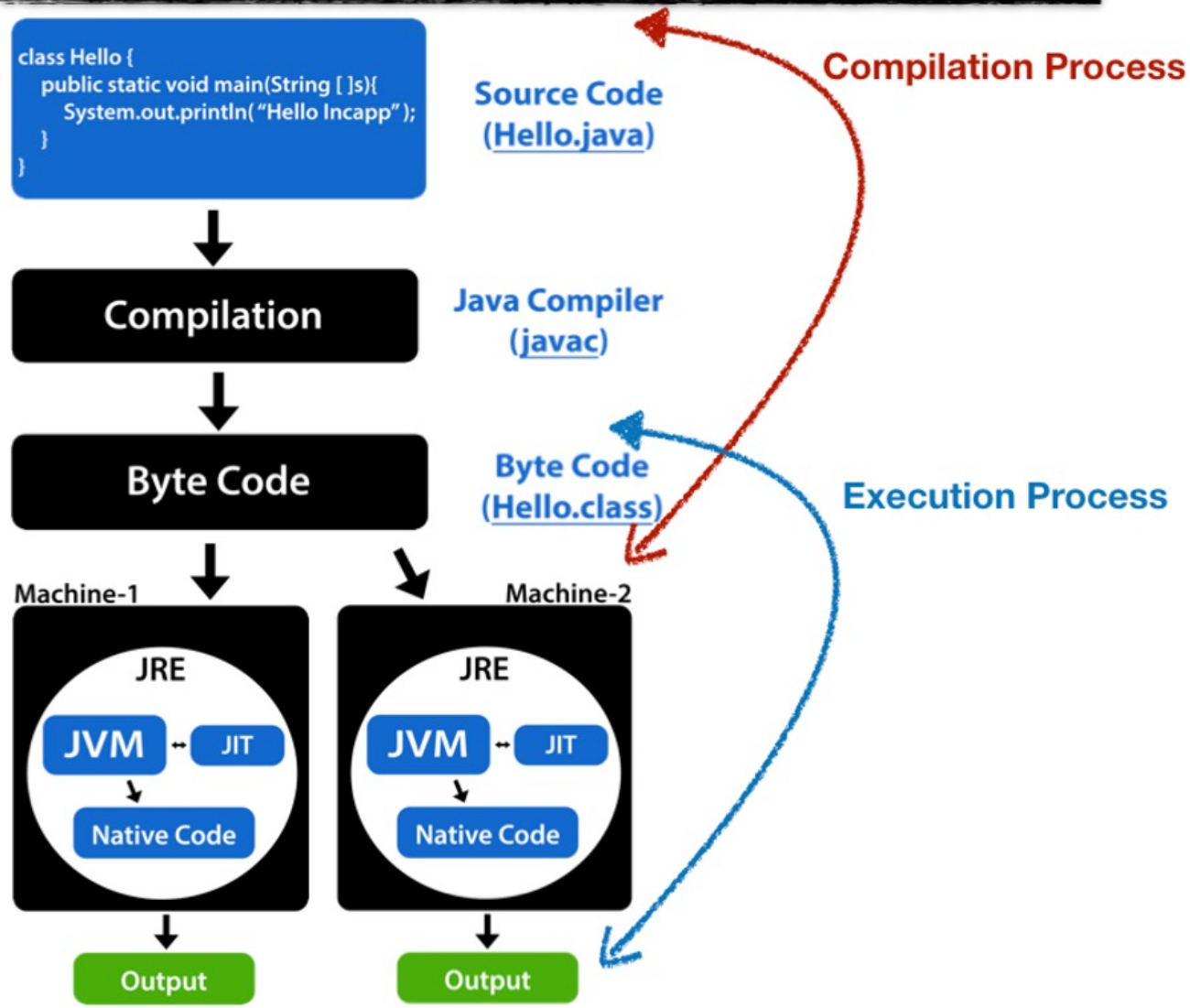
- Initiated in June 1991 and First version came in 1995.
- Developed by Sun Microsystems.
- Best suited for Security and Platform Independency.
- First name: Oak
- Second name: Green
- Final name: Java (came from Java Coffee of Indonesia).
- Syntax came form C and C++
- Oracle Corporation acquired Sun Microsystems in 2010.

Java Applications

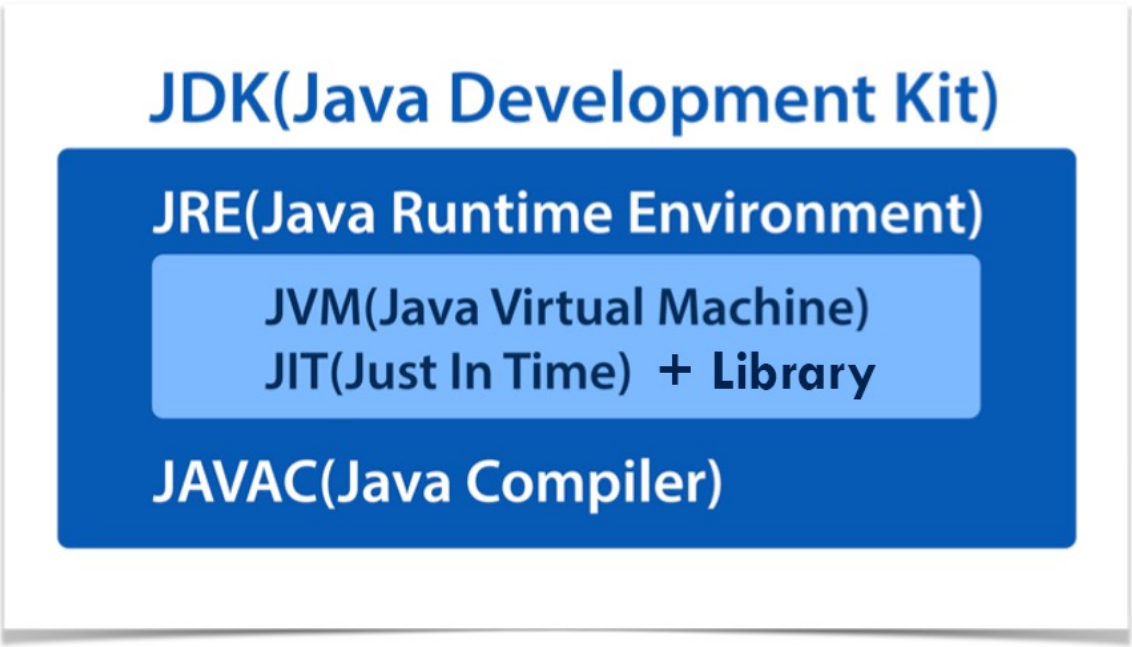
Java Facts

- MORE THAN **95% OF THE ENTERPRISES ARE** USING JAVA.
- MORE THAN **3 BILLION DEVICES** ARE USING JAVA.
- MORE THAN **60 BILLION JVMs** ARE RUNNING.
- JAVA IS **#1 CHOICE** FOR BACKEND DEVELOPMENT.

Java Code Compilation & Execution Process

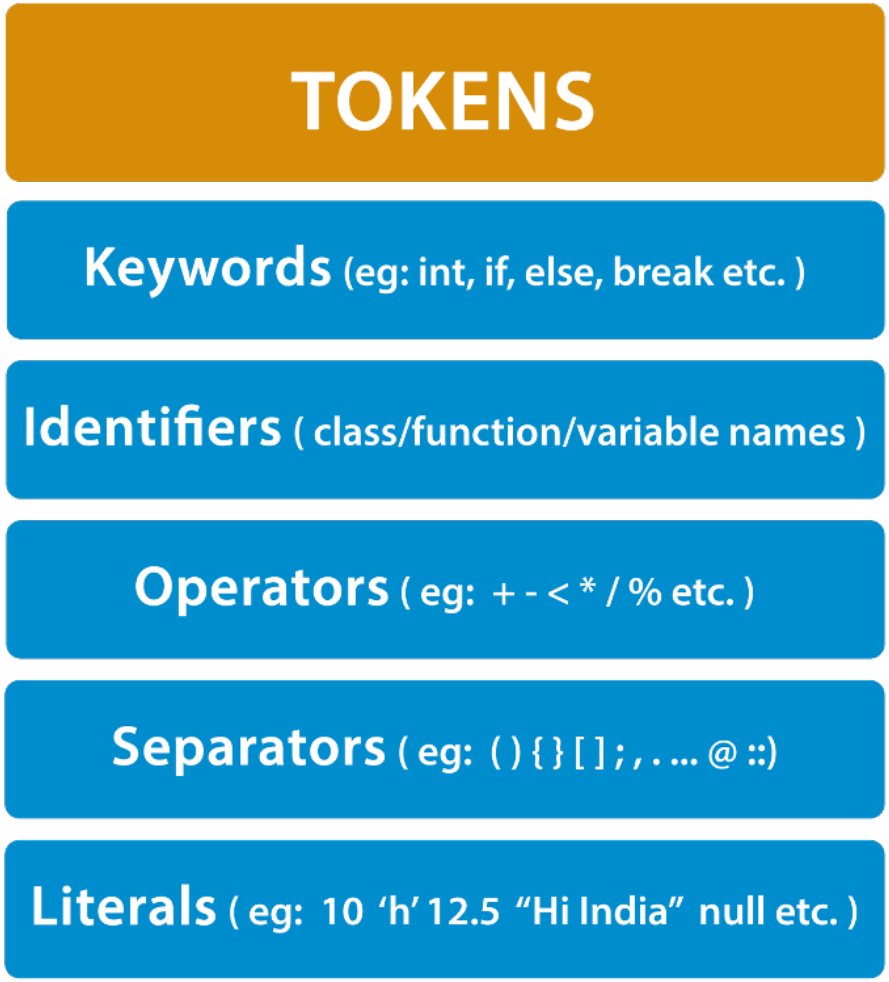


What is JRE / JVM / JDK ?



- **JRE** is an environment where the java's byte code execute.
- **JVM** is the machine that executes the java's byte code.
- **JIT(Just-In Time)** is an interpreter used by JVM to interpret java's byte code to native code(machine code) that gives output.
- **JAVAC** is the compiler that compiles the source code and generates byte code.
- **JDK** is the kit which contains the JRE, JVM and JAVAC.

Tokens



Every Individual units in a program is TOKEN

Keyword Token

A word created by language for particular job that can not be changed by programmer is a keyword.

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello INCAPP");
        int num1=10, num2=20;
        int result=num1+num2;
        System.out.println("Sum: "+result);
    }
}
```

Keywords in java

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

The **goto** and **const** keywords are not in use.
true, false, null are the reserved words not keyword.

Identifier Token

Identifier is a word created by a programmer for variable-name , function-name, class-name and interface-name to use them further.

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello INCAPP");
        int num1=10, num2=20;
        int result=num1+num2;
        System.out.println("Sum: "+result);
    }
}
```

Rules for Identifiers

Rules:

1. It can only be from Alphabets(a-z, A-Z) and Digit(0-9) and underscore(_) and dollar ($).
2. It can not start from digit.
3. Only Underscore is not allowed.
4. Keyword can not be an Identifier.

Operator Token

Operator is a symbol used to do operation on data.

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello INCAPP");
        int num1=10, num2=20;
        int result=num1+num2;
        System.out.println("Sum: "+result);
    }
}
```

List of Operators

Assignment	=
Arithmetic	+ - * / %
Shorthand/Compound	+= -= *= /= %=
Increment-Decrement	++ --
Relational	< <= > >= == !=
Logical	&& || !
Conditional	?:
Bitwise	& | ^ >> << ~

Separator Token

Separator is a symbol that is used to do separation in code.

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello INCAPP");
        int num1=10, num2=20;
        int result=num1+num2;
        System.out.println("Sum: "+result);
    }
}
```

List of Separators

( )	Round Bracket
{ }	Curly Bracket
[ ]	Square Bracket
;	Semicolon
,	Coma
.	Dot
...	Triple Dot
@	At
:	Colon
::	Double Colon

Literal Token

Literal is the constant data in a program.

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello INCAPP");
        int num1=10  num2=20
        int result=num1+num2;
        System.out.println("Sum: "+result);
    }
}
```

Literals

Integer Literals
(20, -90, 0b10101, 012, 0x23a1, 2_124)

Floating Point Literals
(2.0, -90.2, 3.45, -2.678, 2_3.4_21)

Character Literals
('h', '2', '@', 'g', '$', '+' , '\t', '\\', '\u0068')

String Literals
("Hi", "12", "Incapp", "@", "+", "a")

Boolean Literals
(true, false)

Null Literal
(null)

Gapping Rule of Tokens

- In between the
  keyword and **keyword**
  identifier and **identifier**
  keyword and **identifier**
  identifier and **keyword**
  gap(space or new line) is compulsory.

- In between other tokens gap is optional.

Data & Variable

Variable is just a container that contains data.

Data

Data in Variable

101	int roll= 101 ;
Rahul	String name= “Rahul” ;
5.9	double height= 5.9 ;

Data Type

Data type is just a word used to specify the type of the data, size of the data and range of the data. Java has two types of Data Types: **Primitive** and **Non-Primitive .**

Primitive(Pre-Defined) Data Types

- byte
- short
- int
- long
- float
- double
- boolean
- char
- void

Non-Primitive(User-Defined) or (Object Type) Data Types

- Array
- Class
- Interface
- Enum

Primitive Data Types

Data Types	Size	Range	Default Values
byte	1 byte	*-128 to 127*	0
short	2 bytes	*-32768 to 32767*	0
int	4 bytes	*-2147483648 to 2147483647*	0
long	8 bytes	*-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807*	0
float	4 bytes	*1.4E-45 to 3.4028235E38*	0.0f or 0.0F
double	8 bytes	*4.9E-324 to 1.79769313486231570E308*	0.0d or 0.0D
boolean	1 bit	*true or false*	false
char	2 byte	*0 to 65535*	'/u0000'
void	NA	*NA*	NA

Type Casting

Converting the type of data into another data type.

Implicit Typecasting

```
int a=14;
System.out.println(a);

double b=a;

System.out.println(b);
```

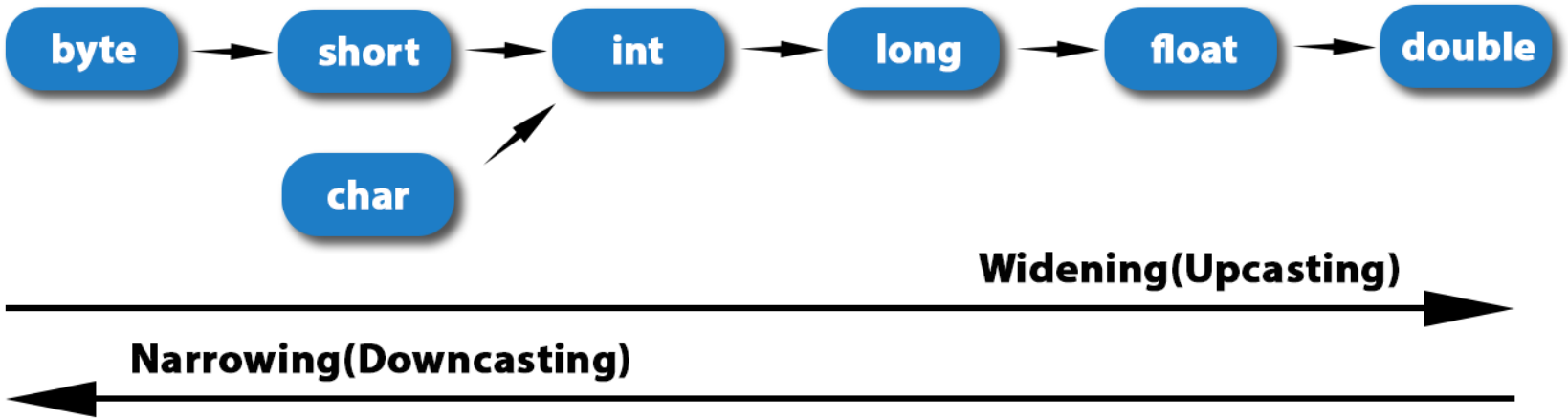
Upcasting (Widening)

Explicit Typecasting

```
double a=14.2;
System.out.println(a);

//int b=a;

int b= (int) a;

System.out.println(b);
```

Downcasting (Narrowing)

TypeCasting Graph



Operators

Operator is a symbol used to do operation on data.

Assignment	**=**
Arithmetic	**+ - * / %**
Shorthand/Compound	**+= -= *= /= %=**
Increment-Decrement	**++ --**
Relational	**< <= > >= == !=**
Logical	**&& || !**
Conditional	**?:**
Bitwise	**& | ^ >> << ~**

Assignment Operator (=)

```
int  a;
a = 10;
```

```
10 = a;
```

```
int  a = 10;
```

```
int  a = 10, b = 15;
```

```
int  a, b;
a = b = 10;
```

Arithmetic Operator (+ - * / %)

```
int  a;
a = 10 + 2;
SOP(a);
```

```
int  a;
a = 10 - 2;
SOP(a);
```

```
int  a;
a = 10 * 2;
SOP(a);
```

```
int  a;
a = 10 / 2;
SOP(a);
```

```
int  a;
a = 10 % 2;
SOP(a);
```

Shorthand Operator (+= -= *= /= %=)

```
int  a=10;
a = a / 5;
SOP(a);
```

```
int  a=10;
a /= 5;
SOP(a);
```

```
int  a=10;
a =/ 5;
SOP(a);
```

```
int  a=10;
a /  = 5;
SOP(a);
```

```
int  a=10;
a = a + 5;
SOP(a);
```

```
int  a=10;
a += 5;
SOP(a);
```

Increment-Decrement Operator (++ --)

```
int a=10;
a = a + 1;
SOP(a);
```

```
int a=10;
a ++;
SOP(a);
```

```
int a=10;
a = a - 1;
SOP(a);
```

```
int a=10;
a --;
SOP(a);
```

Postfix

```
int a=10;
a = a + 1;
SOP(a);
```

```
int a=10;
++ a;
SOP(a);
```

```
int a=10;
a = a - 1;
SOP(a);
```

```
int a=10;
-- a;
SOP(a);
```

Prefix

```
10 ++ ;
```

```
int a=10;
a + + ;
SOP(a);
```

Relational Operator (< <= > >= == !=)

```
int  a=10,b=5;
boolean c=a<b;
SOP(c);
```

```
int  a=10,b=5;
boolean c=a==b;
SOP(c);
```

```
int  a=10,b=5;
boolean c=a< =b;
SOP(c);
```

```
int  a=10,b=5;
boolean c=a<=b;
SOP(c);
```

```
int  a=10,b=5;
boolean c=a!=b;
SOP(c);
```

```
int  a=10,b=5;
boolean c=a>b;
SOP(c);
```

```
int  a=10,b=5;
boolean c=a>=b;
SOP(c);
```

Logical Operator (&& || !)

true && true	true
true && false	false
false && true	false
false && false	false
true || true	true
true || false	true
false || true	true
false || false	false
! true	false
! false	true

```
int  a=10,b=5;
boolean c=a>b && b<3;
SOP(c);
```

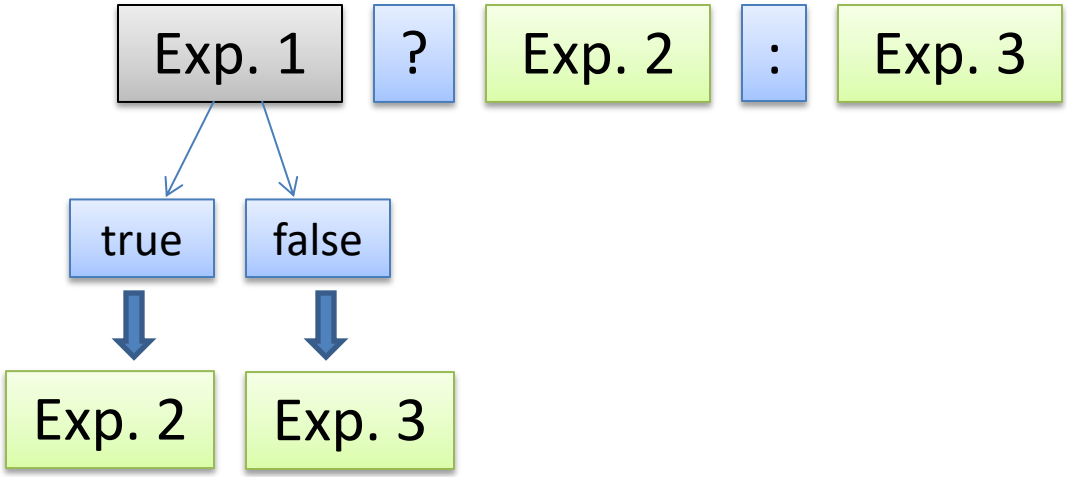
```
int  a=10,b=5;
boolean c=a<b && b<3;
SOP(c);
```

```
int  a=10,b=5;
boolean c=a<b || b<3;
SOP(c);
```

```
int  a=10,b=5;
boolean c=a>b || b==3;
SOP(c);
```

```
int  a=10,b=5;
boolean c=!(a>b) || !(b==3);
SOP(c);
```

Conditional Operator (?:)




```
int  a=10,b=5;
int c=a<b ? a+12/3-2 :  b%5+a-7;
SOP(c);
```

Bitwise Operator (& | ^ >> << ~)

```
int  a=12,b=5;
int c=a & b;
SOP(c);
```

000....001100
000....000101

000....000100

```
int  a=12,b=5;
int c=a | b;
SOP(c);
```

000....001100
000....000101

000....001101

```
int  a=12,b=5;
int c=a ^ b;
SOP(c);
```

000....001100
000....000101

000....001001

```
int  a=12;
int c=a >> 1;
SOP(c);
```

000....001100

0000....00110

```
int  a=12;
int c=a << 1;
SOP(c);
```

000....001100

000....011000

```
int  a=12;
int c= ~ a;
SOP(c);
```

~a = - (a+1)

incapp
The CODING institute

Printing Functions in java

printf: print with format specifier

format: print with format specifier (same as printf)

print: print without format specifier

println: print with new line and without format specifier

```java
public class Test {
    public static void main(String[] args) {
        double x=12.6135;
        System.out.printf("%f\n",x); // 12.613500
        System.out.format("%f\n",x); // 12.613500
        System.out.print(x+"\n");    // 12.6135
        System.out.println(x);       // 12.6135
    }
}
```

```
<terminated> Test [Java Application]
12.613500
12.613500
12.6135
12.6135
```

Format Specifiers in Java

incapp
The CODING institute

- **%d, %o, %x, %X -** Integer
- **%f -** Floating Point
- **%e, %E -** Exponential (Scientific Notation)
- **%s, %S -** String
- **%c, %C -** Character
- **%b, %B -** Boolean
- **%n -** New Line

```java
public class Test {
    public static void main(String[] args) {
        int a=10;
        double b=12.6;
        String c="Incapp";
        char d='h';
        boolean e=true;
        System.out.printf(" %d%n %f%n %s%n %c%n %b%n ",a,b,c,d,e);
    }
}
```

```
<terminated> Test [Java Application] /
 10
 12.600000
 Incapp
 h
 true
```

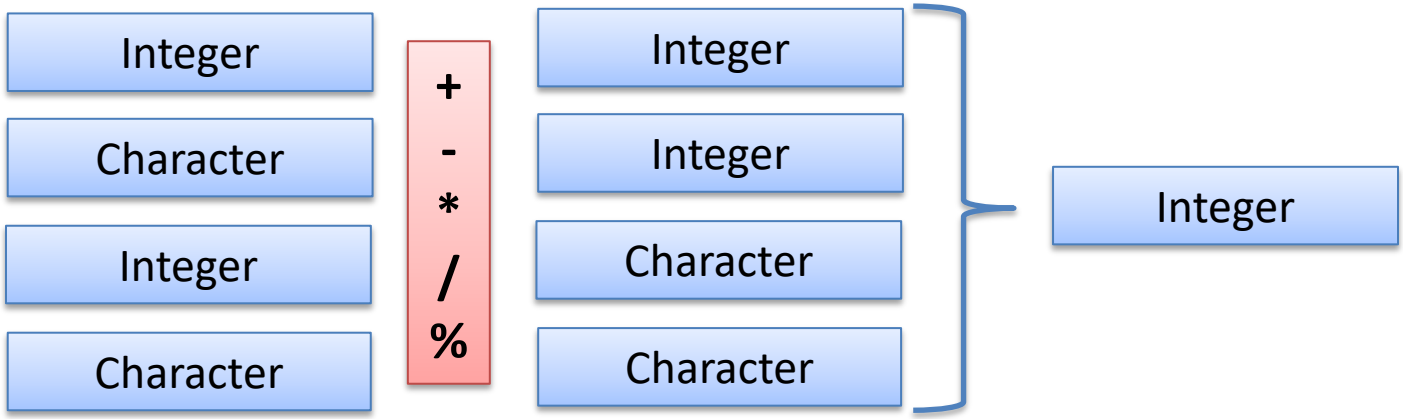
```
public class Test {
    public static void main(String[] args) {
        int a=14;                                   //OUTPUTS
        System.out.printf("%d%n",a);                //14
        System.out.printf("%o%n",a);                //16
        System.out.printf("%x%n",a);                //e
        System.out.printf("%X%n",a);                //E
        System.out.printf("%10d Hi%n",a);           //        14 Hi
        System.out.printf("%-10d Hi%n",a);          //14         Hi

        double b=12.3152;
        System.out.printf("%f%n",b);                //12.315200
        System.out.printf("%.2f%n",b);              //12.32
        System.out.printf("%e%n",b);                //1.231520e+01
        System.out.printf("%E%n",b);                //1.231520E+01

        boolean c=true;
        System.out.printf("%b%n",c);                //true
        System.out.printf("%B%n",c);                //TRUE

        char d='h';
        System.out.printf("%c%n",d);                //h
        System.out.printf("%C%n",d);                //H

        String e="Incapp";
        System.out.printf("%s%n",e);                //Incapp
        System.out.printf("%S%n",e);                //INCAPP
        System.out.printf("%.2s%n",e);              //In
        System.out.printf("'%-10s'%n",e);           //'Incapp    '
        System.out.printf("'%10s'%n",e);            //'    Incapp'
    }
}
```

Scanning data from Keybord

```
java.util.Scanner  sc=new java.util. Scanner( System.in );

int age =sc.nextInt();
double height =sc.nextDouble();
String name =sc.next();
```

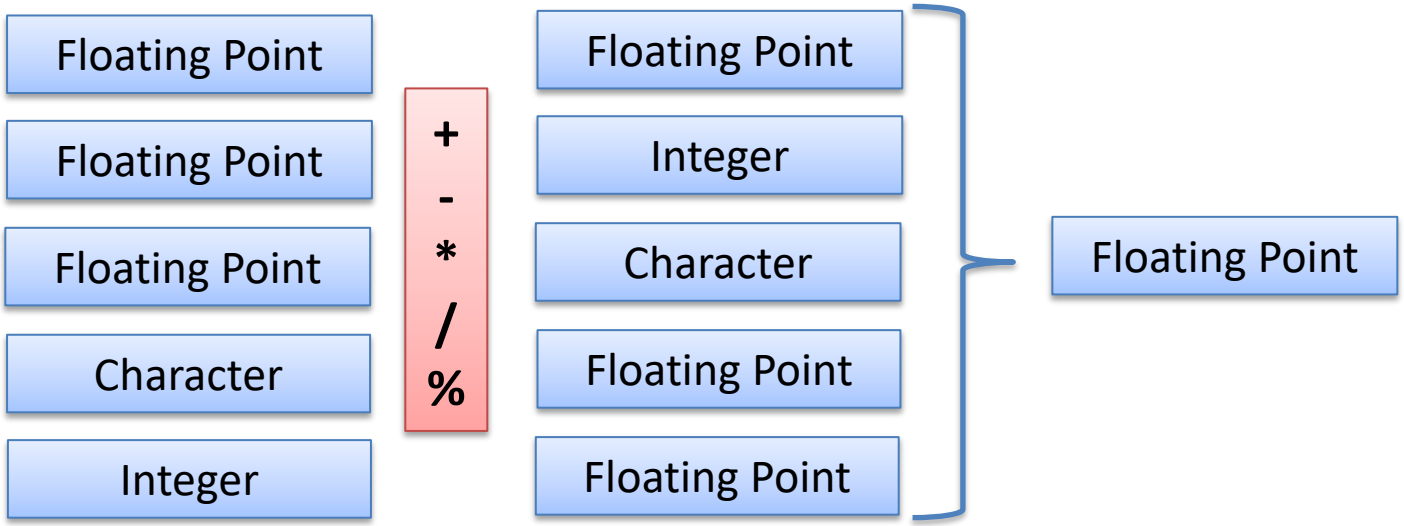
Expression Rules

Println function is Expression based. So we have to understand expression rules.



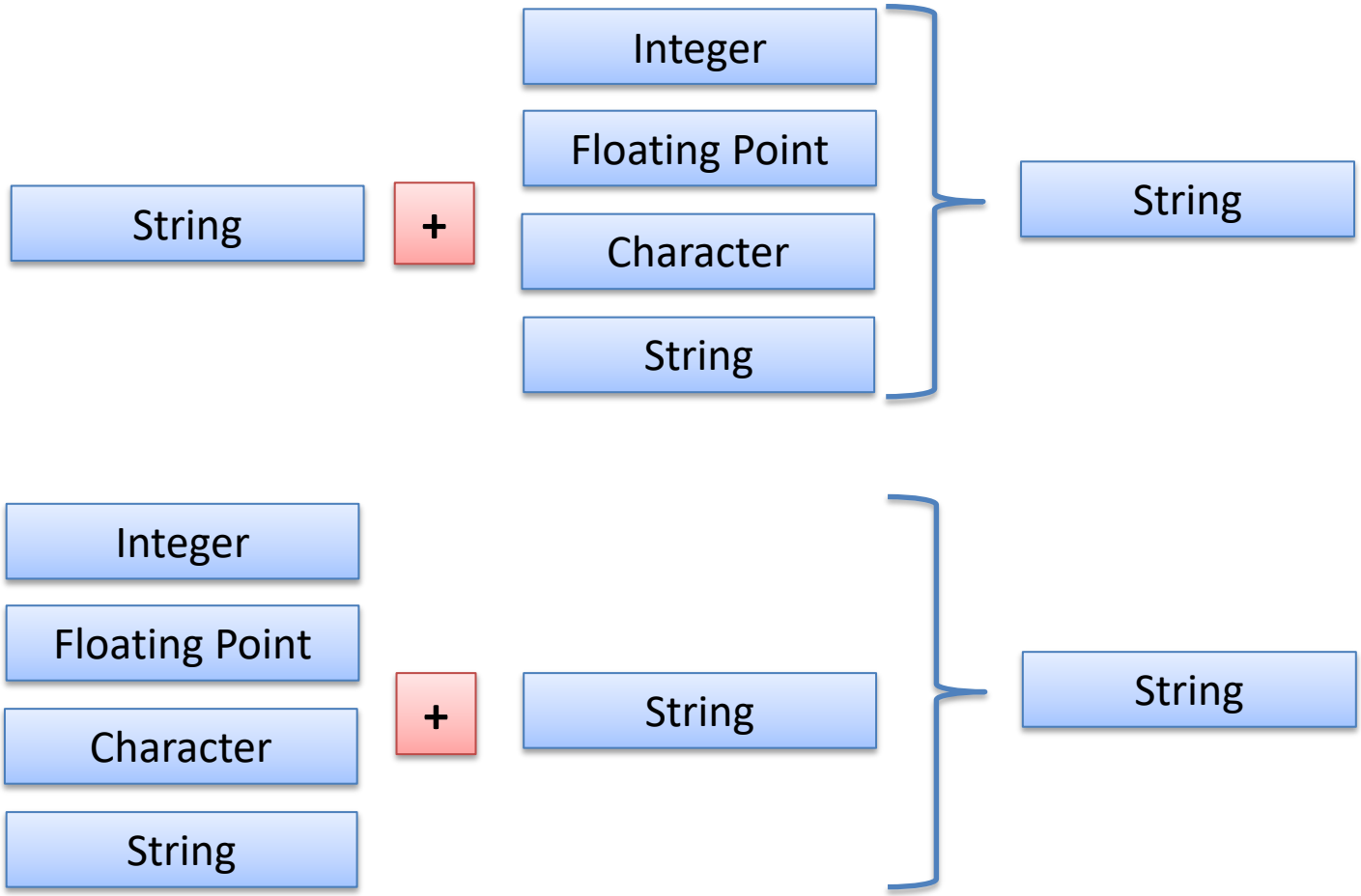
For Example:
7+2 = 9 7/2 = 3 (not 3.5) 'a'+10 = 107 10+ 'a' = 107 'h'*10 = 1040

Expression Rules



For Example:
7.0+2 = 9.0 7/2.0 = 3.5 ‘a’+10.1=107.1 10.0+ ‘a’=107.0 ‘h’*10.0=1040.0

Expression Rules

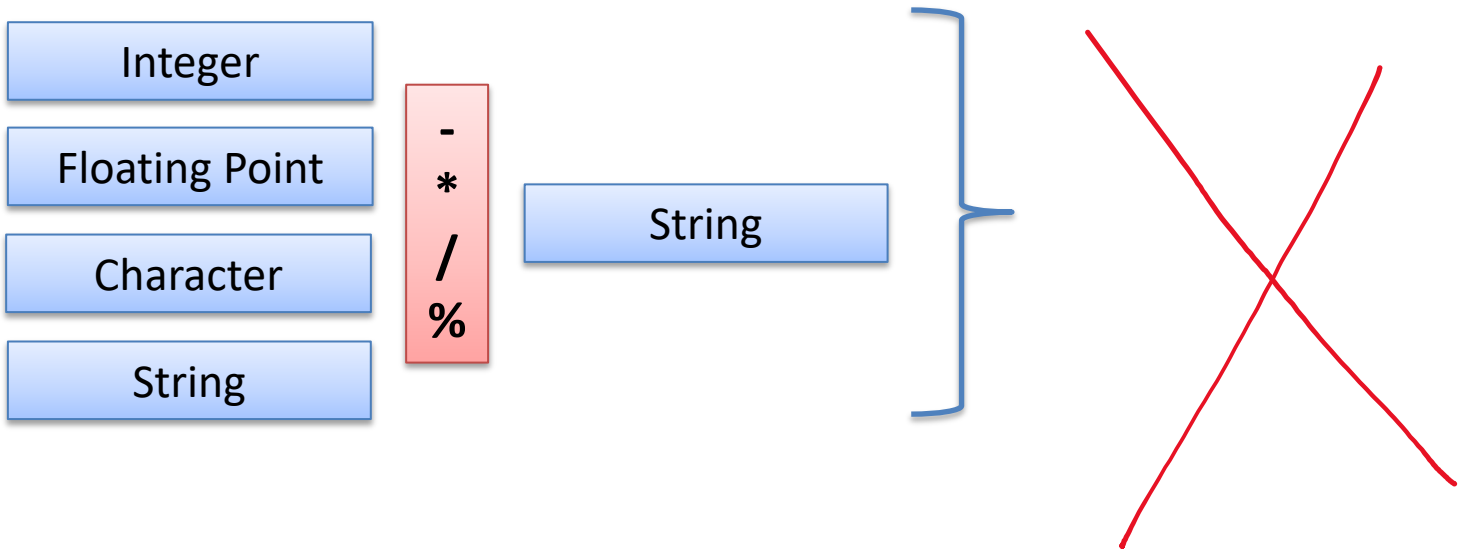


Expression Rules

For Example:

5 + “hi” = “5hi”
“hi”+5 = “hi5”;
4+2+ “hi” = “6hi”
5.8+ “hi” = “5.8hi”;
5+6+ “hi” + 5+6= “11hi56”
“hello”+ “hi” = “hellohi”
“hello”+ ‘a’= “helloa” (not hello97)
‘a’ + “hello” = “ahello”

Expression Rules



Decision Making

- Decision Making is used to execute a block of code on condition, if the condition is satisfied then code will execute otherwise not.
- Types of decision making:

if	If else	if else if	switch

if statement

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

Here, expression must be boolean.
Boolean expression returns either true or false.

Expression -> **true**

```
int age=25;
if (age>18) {
  // statements
}
// statements after if
```

Expression -> **false**

```
int age=15;
if (age>18) {
  // statements
}
// statements after if
```

If-else statement

incapp The CODING institute

Expression -> **true**

```
int age=25;
if (age>18) {
  // statements of if
}
else {
   // statements of else
}
// statements after if else
```

Expression -> **false**

```
int age=15;
if (age>18) {
  // statements of if
}
else {
  // statements of else
}
// statements after if else
```

If-else-if statement

```
int age=15;
if (age>60) {
   // statements of if
}
else if (age>18) {
   // statements of else  if
}
else {
  // statements of else
}
// statements after if else if
```

Switch statement

```
switch (value/expression) {
  case value1 :
    // statements of case
    break;
  case value2 :
    // statements of case
    break;
  default :
    // statements of default
}
// statements after switch
```

- Switch works with the byte, short, char, and int, String class, and few wrapper classes Byte, Short, Character and Integer.
- Switch does not work with float, double, boolean and long.

Loop Controls

- Loop is used to execute a block of code for multiple times.
- Write the code once and execute it many times.
- Types of loop:

for | **while** | **do while** | **for-each(enhanced for)**

For Loop

incapp The CODING institute

Without Loop

```
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
```

With Loop

```
for ( int a=1 ; a<=10 ; a++ ) {

   SOP("Rahul");

}
```

```
for ( Initialization ; TestExpression ; Update )
{
    // statements
}
```

While Loop

incapp The CODING institute

Without Loop

```
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
SOP("Rahul");
```

With Loop

```
int a=1;
while ( a<=5 ) {
   SOP("Rahul");
   a++;
}
```

```
while ( TestExpression) {
    // statements
}
```

Do-While Loop

incapp
The CODING institute

Without Loop
SOP("Rahul"); SOP("Rahul"); SOP("Rahul"); SOP("Rahul"); SOP("Rahul");

With Loop

int a=1;
do {
  SOP("Rahul");
  a++;
} **while** (**a<=5**) ;

```
do {
    // statements
} while ( TestExpression) ;
```

Break keyword

```
for( init; Test-Exp; Update){
  // statements
  if (BreakCondition) {
    // statements
    break;
  }
  // statements
}
```

```
do{
  // statements
  if (BreakCondition) {
    // statements
    break;
  }
  // statements
} while (Test-Exp);
```

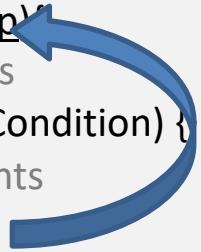
```
while (Test-Exp){
  // statements
  if (BreakCondition) {
    // statements
    break;
  }
  // statements
}
```

Continue keyword

```
for( init; Test-Exp; Update){
  // statements
  if (ContinueCondition) {
    // statements
    continue;
  }
  // statements
}
```

```
do{
  // statements
  if (ContinueCondition) {
    // statements
    continue;
  }
  // statements
} while (Test-Exp);
```

```
while (Test-Exp){
  // statements
  if (ContinueCondition) {
    // statements
    continue;
  }
  // statements
}
```

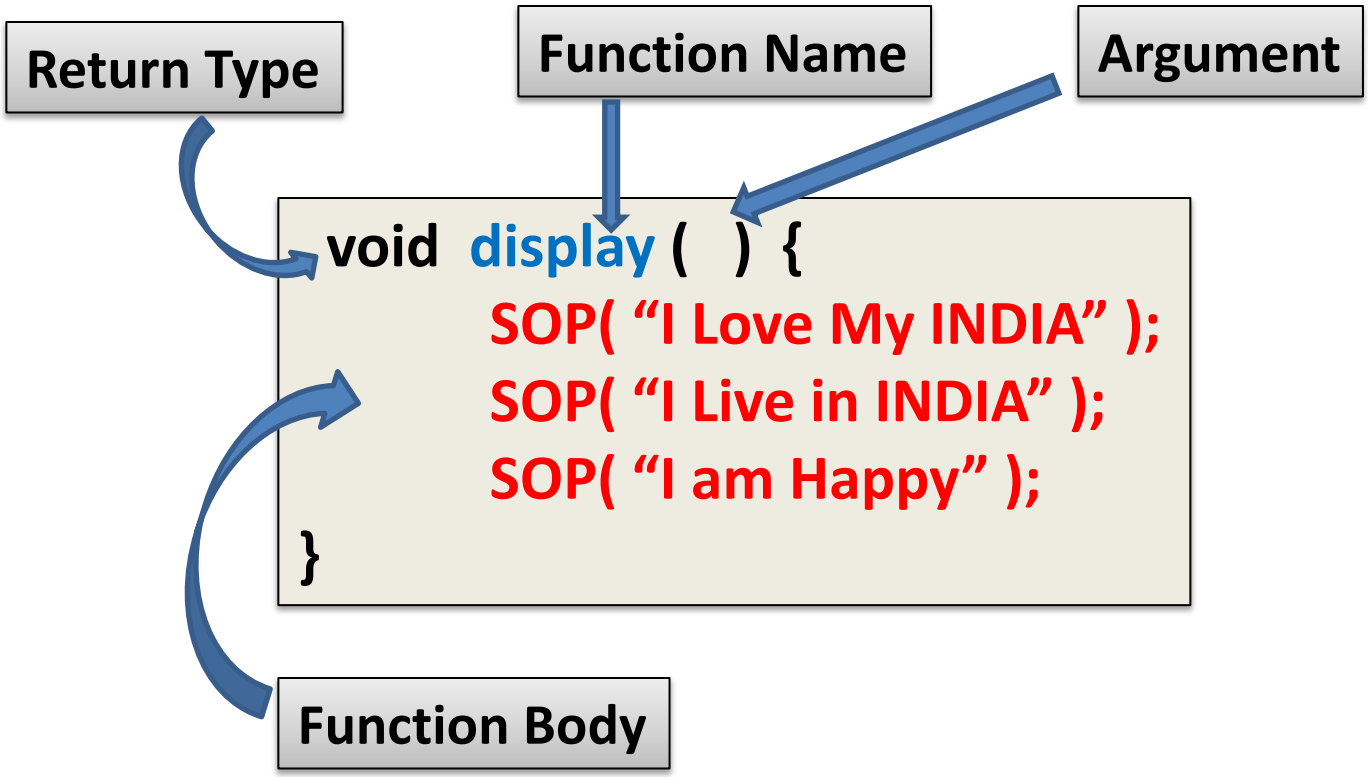


Function/Method

```
PSVM(-){
        SOP( “Hello” );
        SOP( “I Love My INDIA” );
        SOP( “I Live in INDIA” );
        SOP( “I am Happy” );
        SOP( “Hi” );
        SOP( “I Love My INDIA” );
        SOP( “I Live in INDIA” );
        SOP( “I am Happy” );
        SOP( “Bye” );
        SOP( “I Love My INDIA” );
        SOP( “I Live in INDIA” );
        SOP( “I am Happy” );
}
```

```
PS  void display(){
        SOP( “I Love My INDIA” );
        SOP( “I Live in INDIA” );
        SOP( “I am Happy” );
}
PSVM(-){
        SOP( “Hello” );
         display();
        SOP( “Hi” );
         display();
        SOP( “Bye” );
        display();
}
```

Function/Method Syntax



Function/Method

Types of Functions:

- Function with No Arguments and No Return Type
- Function with Arguments but No Return Type
- Function with No Arguments but Return Type
- Function with Arguments and Return Type
- Function with simple Return keyword

No Arguments and No Return Type

```
public class Demo{
        public static  void add(){
                int a=10,b=20,r=a+b;
                SOP( “Sum: ”+r );
        }
        public static void main(String[] s){
                 add();
        }
}
```

Arguments and No Return Type

```
public class Demo{
	public static  void add(int a, int b){
		int r= a+b;
		SOP( “Sum: ”+r );
	}
	public static void main(String[] s){
		 add(12,34);
	}
}
```

No Arguments but Return Type

```
public class Demo{
	public static  int add(){
		int a=10,b=20,r=a+b;
		return r;
	}
	public static void main(String[] s){
		int r=add();
		SOP( “Sum: ”+r );
	}
}
```

Arguments but Return Type

```
public class Demo{
	public static int add(int a, int b){
		int r= a+b;
		 return r;
	}
	public static void main(String[] s){
		int r=add(12,34);
		SOP( “Sum: ”+r );
	}
}
```

Simple Return keyword

```
public class Demo{
	public static void xyz(int a, int b){
		SOP( “Hello” );
		if(a==b){
			return;
		}
		SOP( “Hi” );
	}
	public static void main(String[] s){
		xyz(10,20);
	}
}
```