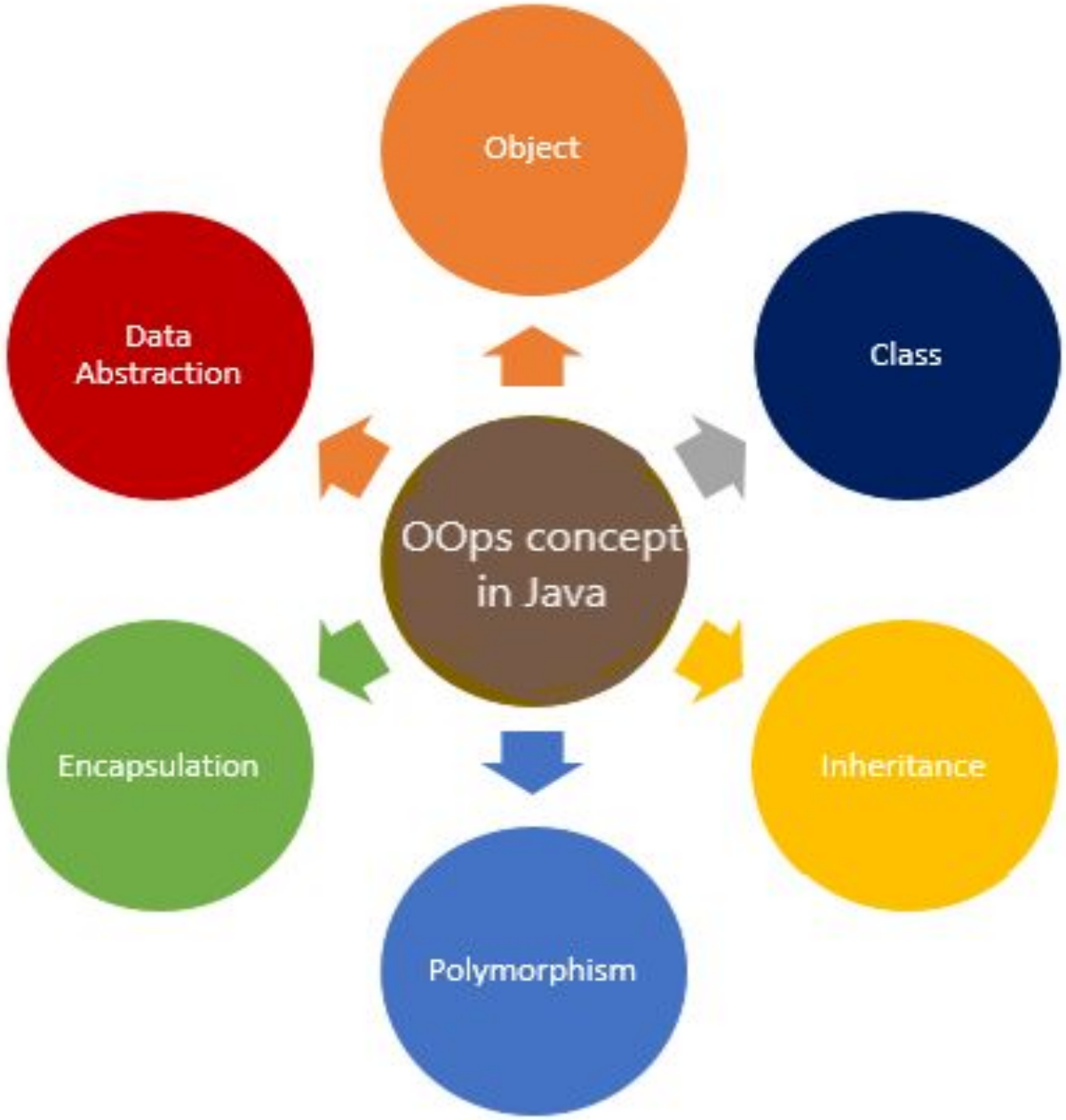


# CS20004: Object Oriented Programming using Java

**Lec-6**

# In this Discussion . . .

- Operators
  - Operator precedence
- Java Expressions
- Selection Statements
- Switch Statements
- References

# Operators

- Operators are used to build value expressions
  - Unary
  - Binary
  - Ternary
- **Arithmetic** Operator
  - +, -, *, /, %
- **Relational** Operator
  - Outcome is always a value of type Boolean
  - ==, ≠ , < , ≤ , > , ≥

# Operators

- **Logical** Operator
  - Logical operators act upon Boolean operands only
  - &, |, !, ∧
- **Short Circuit Logical** Operator
  - If these operators are used, java will not evaluate the second operand if the result can be determined by the first operand alone
  - & &, ||

# Operators

- **Increment & Decrement** Operator
  - The operand must be a numerical variable
  - prefix version evaluates the value of the operand after performing the increment/decrement operation
  - postfix version evaluates the value of the operand before performing the increment/decrement operation
- **Bitwise Logical** Operator
  - &, |, ~, ⋀, < <, > >, > > >

# Operators

- **Assignment** Operator
  - Types of the variable and expression must be compatible
  - =
- Other Operators
  - Conditional operator (? :)
  - [ ]
  - (params)
  - (type)
  - new
  - instanceof
  - .

# Unary Operators Examples

```
public class Unaryoperatorsexp
{
        public static void main(String args[])
        {
                int x=10;
                System.out.println(x++);

                System.out.println(++x);

                System.out.println(x--);

                System.out.println(--x);

                int a = 10;
                int b = 10;

                System.out.println(a++ + ++a);
                System.out.println(b++ + b++);


        }
}
```

# Unary Operators Examples

```
public class Unaryoperatorsexp
{
        public static void main(String args[])
        {
                int x=10;
                System.out.println(x++);          10

                System.out.println(++x);          12

                System.out.println(x--);          12

                System.out.println(--x);          10

                int a = 10;
                int b = 10;
                System.out.println(a++ + ++a);    22
                System.out.println(b++ + b++);    21
        }
}
```

# Arithmetic Operators Example

```
public class Arithmeticopexps
{
        public static void main(String args[])
        {
                int a=10;
                int b=5;
                System.out.println(a+b);

                System.out.println(a-b);

                System.out.println(a*b);

                System.out.println(a/b);

                System.out.println(a%b);
        }
}
```

# Unary Operators Examples

```
public class Arithmeticopexps
{
        public static void main(String args[])
        {
                int a=10;
                int b=5;

                System.out.println(a+b);

                System.out.println(a-b);

                System.out.println(a*b);

                System.out.println(a/b);

                System.out.println(a%b);
        }
}
```

15

5

50

2

0

# Left shift & Right Shift Operators

- The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.
- The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

# Left shift & Right Shift Operators

```
public class Shiftopexp
{
	public static void main(String args[])
	{
		System.out.println(10<<2);

		System.out.println(10<<3);

		System.out.println(20<<2);

		System.out.println(15<<4);

		System.out.println(10>>2);

		System.out.println(20>>2);

		System.out.println(20>>3);
	}
}
```

# Left shift & Right Shift Operators

```
public class Shiftopexp
{
        public static void main(String args[ ])
        {
                System.out.println(10<<2);
                System.out.println(10<<3);
                System.out.println(20<<2);
                System.out.println(15<<4);
                System.out.println(10>>2);
                System.out.println(20>>2);
                System.out.println(20>>3);
        }
}
```

| | |
|---|---|
| → | 40 ($10 * 2^2$) |
| → | 80 ($10 * 2^3$) |
| → | 80 ($20 * 2^2$) |
| → | 240 ($15 * 2^4$) |
| → | 2 ($10 / 2^2$) |
| → | 5 ($20 / 2^2$) |
| → | 2 ($20 / 2^3$) |

# AND Operator: Logical && vs Bitwise &

- The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.
- The bitwise & operator always checks both conditions whether first condition is true or false.

# AND Operator: Logical && vs Bitwise &

```
public class Logicalopexp
{
	public static void main(String args[])
	{
		int a=10;
		int b=5;
		int c=20;

		System.out.println(a< b && a++ < c);

		System.out.println(a);

		System.out.println(a < b & a++ < c);

		System.out.println(a);
	}
}
```

# AND Operator: Logical && vs Bitwise &

```
public class Logicalopexp
{
        public static void main(String args[])
        {
                int a=10;
                int b=5;
                int c=20;

                System.out.println(a < b && a++ < c);

                System.out.println(a);

                System.out.println(a < b & a++ < c);

                System.out.println(a);
        }
}
```

false (false && true )

10 (as second condition is not checked )

false (false & true )

11 (because second condition is checked)

# Operator Precedence

- The operator precedence represents how two expressions are bind together.
- In an expression, it determines the grouping of operators with operands and decides how an expression will evaluate.
- While solving an expression two things must be kept in mind the first is a precedence and the second is associativity.
- When operators have the same precedence, the earlier one binds stronger

# Operator Precedence (Highest to Lowest: From Top to Bottom)

| Operator Type | Category | Precedence |
|---|---|---|
| Unary | postfix | *expr++ expr--* |
| | prefix | *++expr --expr +expr -expr ~ !* |
| Arithmetic | multiplicative | * / % |
| | additive | + - |
| Shift | shift | << >> >>> |
| Relational | comparison | < > <= >= instanceof |
| | equality | == != |
| Bitwise | bitwise AND | & |
| | bitwise exclusive OR | ^ |
| | bitwise inclusive OR | \| |
| Logical | logical AND | && |
| | logical OR | \|\| |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= *= /= %= &= ^= \|= <<= >>= >>>= |

# Java Expressions

- An expression in Java is a series of operators, variables, and method calls constructed according to the syntax given, the language for evaluating a single value is as follows:
- For instance,

```
int marks;
marks = 90;
```

Here marks=90 is an expression that returns an int value.

Let us take another instance:

```
Double a = 2.2, b = 3.4, result;
result = a + b - 3.4;
```

Here in this example,a+b-3.4 is an expression.

# Simple Expressions

- A simple expression is a literal method call or variable name without any usage of an operator.

  For instance:

```
43                          // integer literal
name                        // variable name
System.out.println("Hello"); // method call
"Java"                      // string literal
133B                        // double precision floating-point literal
32L                         // long integer literal
```

- A simple expression in java has a type that can either be a primitive type or a reference type. In this example, 43 is a 32-bit integer, java is a string, 32L is a long 64-bit integer, etc.

# Compound Expressions

- It generally involves the usage of operators. It comprises one or more simple expressions, which are then integrated into a larger expression by using the operator. Consider another example in order to understand more clearly.

```
Double a = 2.3, b = 3.2, number;
number = a + b - 3.4;
```

# Selection Statements

- Java selection statements allow to control the flow of program's execution based upon conditions known only during run-time.
- Selection statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided.
- There are two types of selections statements in Java, i.e., **if** *statement* and **switch** *statement*.
- if statement syntax:-

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

The **expression** must be of *type* **Boolean**

# Selection Statements

- if-else statements

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

# Selection Statements (if-else statements Example)

```
public class Ifelseexp
{
	public static void main(String[] args)
	{
		int x = 10;
		int y = 12;
		if(x+y < 10)
		{
			System.out.println("Gajar ka Halwa is the best food");
		}
		else
		{
			System.out.println("Pizza from Ovenstory rocks over
Dominos");
		}
	}
}
```

# Selection Statements (if-else statements Example)

```
public class Ifelseexp
{
	public static void main(String[] args)
	{
		int x = 10;
		int y = 12;
		if(x+y < 10)
		{
			System.out.println("Gajar ka Halwa is the best food");
		}
		else
		{
			System.out.println("Pizza from Ovenstory rocks over
Dominos");
		}
	}
}
```

Pizza from Ovenstory rocks over Dominos

# Selection Statements

- if-else-if statements

```
if(expression)
{
      //statements
}
else if (expression 2)
{
     //statements
}
else if (expression 3)
{
    //statements
}
.........
else
{ //statement
}
```

# Selection Statements

- if-else-if statements

```
public class Ifelseifexp
{
	public static void main(String[] args)
	{
		String city = "Delhi";
		if(city == "Meerut")
		{
			System.out.println("Tikki");
		}
		else if (city == "Noida")
		{
			System.out.println("Rajma Chawal");
		}
		else if(city == "Agra")
		{
			System.out.println("Petha");
		}
		else
		{
			System.out.println("Chole Bhature");
		}
	}
}
```

# Selection Statements

- if-else-if statements

```
public class Ifelseifexp
{
	public static void main(String[] args)
	{
		String city = "Delhi";
		if(city == "Meerut")
		{
			System.out.println("Tikki");
		}
		else if (city == "Noida")
		{
			System.out.println("Rajma Chawal");
		}
		else if(city == "Agra")
		{
			System.out.println("Petha");
		}
		else
		{
			System.out.println("Chole Bhature");
		}
	}
}
```

Chole Bhature

# Switch Statement

- switch provides a better alternative than if-else-if when the execution follows several branches depending on the value of an expression

```
switch(expression)
{
    caseValue1: statement1;  break;
    caseValue2: statement2;  break;
    ……..
    default:statement;
}
```

# Switch Statement

- Expression must be of type byte, short, int or char
- Each of the case values must be a literal of the compatible type
- Case values must be unique
- Break makes sure that only the matching statement is executed

```
switch(expression)
{
    caseValue1: statement1;  break;
    caseValue2: statement2;  break;
    ……..
    default:statement;
}
```

# Switch Statement Example

```
public class Switchexp
{
	public static void main(String[] args)
	{
		String city = "Delhi" ;
		switch (city)
		{
			case "Meerut":
				System.out.println("Tikki");
				break;
			case "Noida":
				System.out.println("Rajma
Chawal");
				break;
			default:
				System.out.println("Chole
Bhature");
		}
	}
}
```

# Switch Statement Example

```
public class Switchexp
{
	public static void main(String[] args)
	{
		String city = "Delhi" ;
		switch (city)
		{
			case "Meerut":
				System.out.println("Tikki");
				break;
			case "Noida":
				System.out.println("Rajma
Chawal");
				break;
			default:
				System.out.println("Chole
Bhature");
		}
	}
}
```

Chole Bhature

# References


1. https://www.geeksforgeeks.org/type-conversion-java-examples/
2. https://www.javatpoint.com/scope-of-variables-in-java
3. https://i.stack.imgur.com/Ij3vJ.png
4. https://www.javatpoint.com/control-flow-in-java
5. https://www.scaler.com/topics/expression-in-java/
6.