STEPS IN PROBLEM SOLVING:

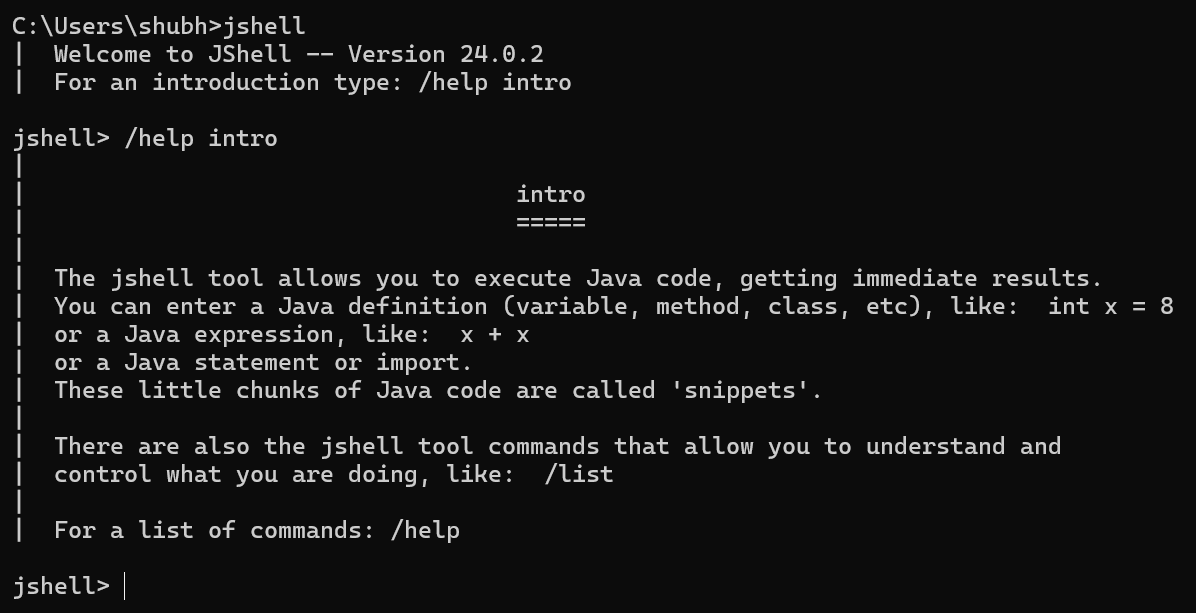
1. Understand the problem.
2. Design
   1. Break the problem down
3. Write your program (and Test)
   1. Express your solution: Language Specific (Syntax)

CONCEPTS:

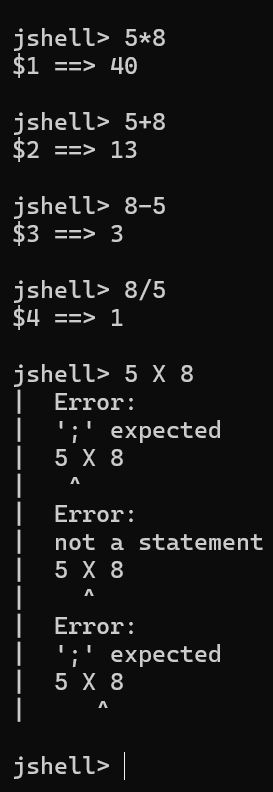
* JShell
* Statements
* Operators
* Operands
* Expressions
* Literals
* Type Conversion
* Operator Precedence
* Variables
* If Statements
* For Loop
* Method/Function

**JShell:**

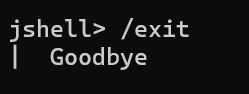
* From Java 9 JShell introduced.
* JAVA REPL (Read, Evaluate, Print, Loop)
* Open command prompt and type command jshell to logged in into JShell.



* JShell gives immediate output. Type in a one line of code and see the output.



* To logged out from JShell type /exit command



**JAVA Statement:**

* A **statement** in Java is a **complete unit of execution**.
* Types of Statements in Java

1. Declaration Statement

Declares a variable or constant.

int a;

double salary;

1. Expression Statement

Performs an action using an expression.

a = 5;

a++;

System.out.println(a);

1. Control Flow Statements

Used to control the flow of execution.

1. Decision-making Statements

if

if-else

switch

Example:

if (a > 10) {

System.out.println("Big");

}

1. Looping Statements

for

while

do-while

Example:

for (int i = 0; i < 5; i++) {

System.out.println(i);

}

1. Jump Statements

break

continue

return

Example:

if (a == 5) break;

* A statement in Java is a complete command that ends with a semicolon (;) or a block { } that tells the JVM to do something.

**JAVA Operators:**

* Predefine operators: \* / + - %

1. Arithmetic Operators (5)

+ (addition)

- (subtraction)

\* (multiplication)

/ (division)

% (modulus)

1. Unary Operators (4 main, but 7 including variations)

+ (unary plus)

- (unary minus)

++ (increment – pre & post)

-- (decrement – pre & post)

! (logical NOT)

~ (bitwise complement)

1. Relational Operators (6)

== (equal to)

!= (not equal to)

> (greater than)

< (less than)

>= (greater than or equal to)

<= (less than or equal to)

1. Logical Operators (3)

&& (logical AND)

|| (logical OR)

! (logical NOT – already listed in unary)

1. Bitwise Operators (7)

& (bitwise AND)

| (bitwise OR)

^ (bitwise XOR)

~ (bitwise NOT – already listed in unary)

<< (left shift)

>> (right shift)

>>> (unsigned right shift)

1. Assignment Operators (12 variations)

= (simple assignment)

+=, -=, \*=, /=, %=

&=, |=, ^=, <<=, >>=, >>>=

1. Conditional (Ternary) Operator (1)

?:

1. Instanceof Operator (1)

Instanceof

**JAVA Operands:**

* Operand: 5 \* 8 (5 and 8 are Operands)

int x = 7 + 3; // 7 and 3 are operands

int y = x \* 2; // x and 2 are operands

if (obj instanceof String) // obj is operand

Arity (How many operands?)

Unary operators → act on one operand

Example: ++x → operand is x

Binary operators → act on two operands

Example: a + b → operands are a and b

Ternary operator → acts on three operands

Example: condition ? value1 : value2 → operands are condition, value1, value2

* **Difference Between Expression & Statement:**

Expression → produces a value

a + b // just a value (not complete action)

Statement → does a complete action

int c = a + b; // statement (assignment action)

* **Operands are the inputs; operators are the actions.**

**JAVA Expressions:**

* **Expressions:** An expression in Java is a combination of operands (variables, literals, values) and operators that evaluates to a single value.
* e.g., int x = 5 + 3;

5 + 3 → expression

It evaluates to 8

The result is assigned to x

* In Short: An **expression** is the **whole formula** that produces a result.

Arithmetic Expression → uses + - \* / %

int result = (a + b) \* c;

Relational Expression → uses == != > < >= <=

boolean check = (a > b);

Logical Expression → uses && || !

boolean valid = (age > 18 && salary > 20000);

Assignment Expression

x = y + 5;

Unary Expression

++a; // increment expression

Ternary Expression

String result = (a > b) ? "A is bigger" : "B is bigger";

Method Call Expression

int len = str.length();

**JAVA Literals:**

* **Literals:** A literal in Java is a constant value that is directly written in the code, without computation.
* It represents a fixed value that remains the same throughout program execution.
* Types of Literals in Java

1. Integer Literals

Whole numbers (no decimal).

Can be written in:

Decimal (base 10) → int a = 100;

Octal (base 8, prefix 0) → int b = 012; // decimal 10

Hexadecimal (base 16, prefix 0x) → int c = 0x1F; // decimal 31

Binary (prefix 0b) → int d = 0b1010; // decimal 10

1. Floating-Point Literals

Numbers with decimal point.

float f = 12.5f; // 'f' is required for float

double d = 45.67; // default is double

1. Character Literals

A single character enclosed in single quotes ' '.

char ch1 = 'A';

char ch2 = '\n’; // escape sequence (newline)

1. String Literals

Sequence of characters enclosed in double quotes " ".

String name = "Java";

1. Boolean Literals

Only two values: true or false.

boolean status = true;

1. Null Literal

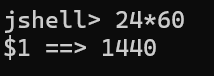
Represents the absence of a value (only for object references).

String str = null;

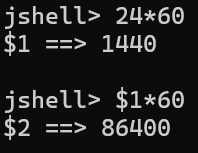
* Literals are fixed values directly written in the program. They are the actual data that variables store.

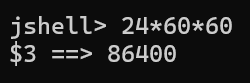
Exercise:

1. Write an expression to calculate number of minutes in a day.

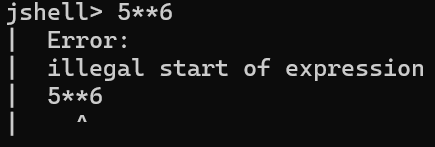


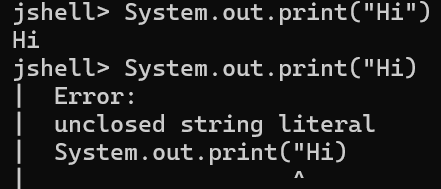
1. Write an expression to calculate number of seconds in a day.





* Some errors which show expression and literals:





**TYPE CONVERSION IN JAVA:**

* Type conversion in Java means **changing a variable from one data type to another**.

1. Widening (Implicit) Conversion / Type Casting
2. Narrowing (Explicit) Conversion / Type Casting
3. Type Promotion in Expressions
4. Conversion Between Objects (Wrapper Classes)
   1. Autoboxing
   2. Unboxing

* Widening (Implicit) Conversion / Type Casting

Also called type promotion.

Happens automatically when a smaller data type is converted to a larger one.

No data loss occurs.

Order of widening (lowest → highest):

byte → short → int → long → float → double

Example:

int num = 10;

double d = num; // int automatically converted to double

System.out.println(d); // 10.0

* Narrowing (Explicit) Conversion / Type Casting

Must be done manually using (type) cast operator.

Converting a larger data type to a smaller one.

Possible data loss or precision loss.

Example:

double d = 10.5;

int num = (int) d; // explicitly cast

System.out.println(num); // 10 (decimal part lost)

* Type Promotion in Expressions

When different types are mixed in an expression, Java promotes smaller types to larger automatically.

Example:

int a = 10;

float b = 5.5f;

float result = a + b; // int promoted to float

System.out.println(result); // 15.5

Another Example:

byte x = 10;

byte y = 20;

int z = x + y; // byte values promoted to int during arithmetic

* Conversion Between Objects (Wrapper Classes)

Java also allows type conversion between primitives and their wrapper classes.

* **Autoboxing**: primitive → wrapper

int a = 5;

Integer obj = a; // auto boxing

* **Unboxing**: wrapper → primitive

Integer obj = 20;

int b = obj; // auto unboxing

* Java automatically performs **widening conversions** (safe), but for **narrowing conversions** you must explicitly cast, as it may lose data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **From Type** | **To Type** | **Conversion Type** | **Example** | **Notes** |
| byte | short, int, long, float, double | **Widening (Implicit)** | byte b = 10; int i = b; | Safe, no data loss |
| short | int, long, float, double | **Widening (Implicit)** | short s = 50; double d = s; | Safe |
| char | int, long, float, double | **Widening (Implicit)** | char c = 'A'; int i = c; // 65 | Converts to Unicode value |
| int | long, float, double | **Widening (Implicit)** | int x = 100; double d = x; | Safe |
| long | float, double | **Widening (Implicit)** | long l = 12345L; float f = l; | Possible precision loss in float |
| float | double | **Widening (Implicit)** | float f = 10.5f; double d = f; | Safe |
| double | float, long, int, short, byte, char | **Narrowing (Explicit)** | double d = 99.99; int i = (int) d; | Decimal part lost |
| float | long, int, short, byte, char | **Narrowing (Explicit)** | float f = 7.6f; int i = (int) f; | Decimal part lost |
| long | int, short, byte, char | **Narrowing (Explicit)** | long l = 1000; int i = (int) l; | Data may overflow |
| int | short, byte, char | **Narrowing (Explicit)** | int i = 130; byte b = (byte) i; | Overflow risk |
| char | byte, short | **Narrowing (Explicit)** | char c = 'A'; byte b = (byte) c; | May lose Unicode value |

| **Example** | **Conversion Rule** |
| --- | --- |
| byte + byte → int | Both promoted to int |
| short + short → int | Both promoted to int |
| char + char → int | Unicode values added → int |
| int + float → float | int promoted to float |
| float + double → double | float promoted to double |

| **Conversion** | **Example** | **Notes** |
| --- | --- | --- |
| Autoboxing (primitive → Wrapper) | int a = 10; Integer obj = a; | Automatic |
| Unboxing (Wrapper → primitive) | Integer obj = 20; int b = obj; | Automatic |

**OPERATOR PRECEDENCE IN JAVA:**

* **Precedence** = which operator is evaluated **first** when an expression has multiple operators.
* **Associativity** = the **direction** of evaluation (left → right or right → left).

| **Precedence** | **Operator(s)** | **Description** | **Associativity** |
| --- | --- | --- | --- |
| 1 (Highest) | (), [], ., :: | Parentheses (grouping), Array index, Member access, Method reference | Left → Right |
| 2 | ++, -- (postfix) | Post-increment/decrement | Left → Right |
| 3 | ++, -- (prefix), + (unary), - (unary), ~, ! | Unary operators | Right → Left |
| 4 | \*, /, % | Multiplication, Division, Modulus | Left → Right |
| 5 | +, - | Addition, Subtraction | Left → Right |
| 6 | <<, >>, >>> | Shift operators | Left → Right |
| 7 | <, <=, >, >=, instanceof | Relational | Left → Right |
| 8 | ==, != | Equality | Left → Right |
| 9 | & | Bitwise AND | Left → Right |
| 10 | ^ | Bitwise XOR | Left → Right |
| 11 | | | Bitwise OR | Left → Right |
| 12 | && | Logical AND | Left → Right |
| 13 | || | Logical OR | Left → Right |
| 14 | ?: | Ternary (conditional) | Right → Left |
| 15 | =, +=, -=, \*=, /=, %= etc. | Assignment operators | Right → Left |
| 16 (Lowest) | , | Comma operator | Left → Right |

Examples:

int a = 10, b = 5, c = 2;

int result1 = a + b \* c; // b\*c first → 10 + 10 = 20

int result2 = (a + b) \* c;// parentheses first → 15 \* 2 = 30

int result3 = a > b && b > c; // > first, then &&

int result4 = a = b = c;// right-to-left → a=2, b=2, c=2

* Shortcut to remember:

Parentheses first,

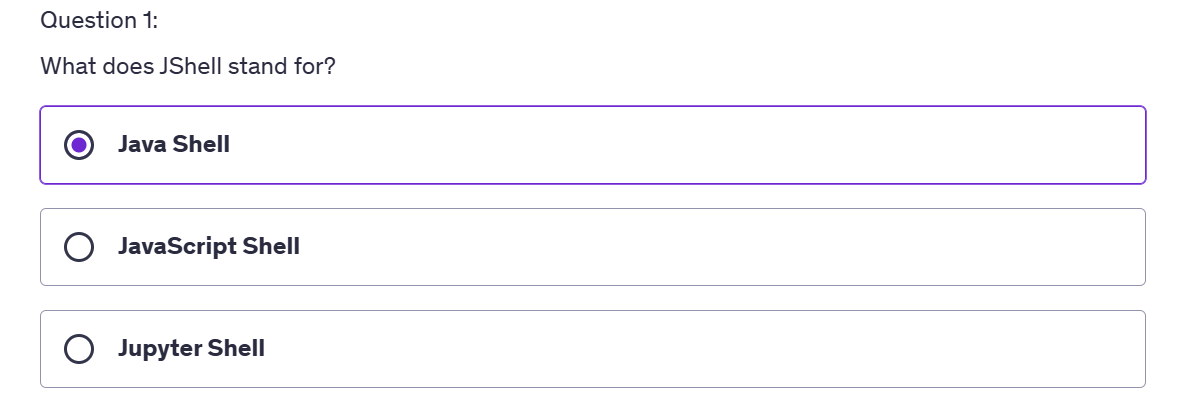
Unary before binary,

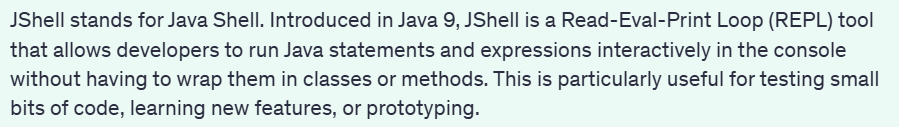
Multiply/Divide before Add/Subtract,

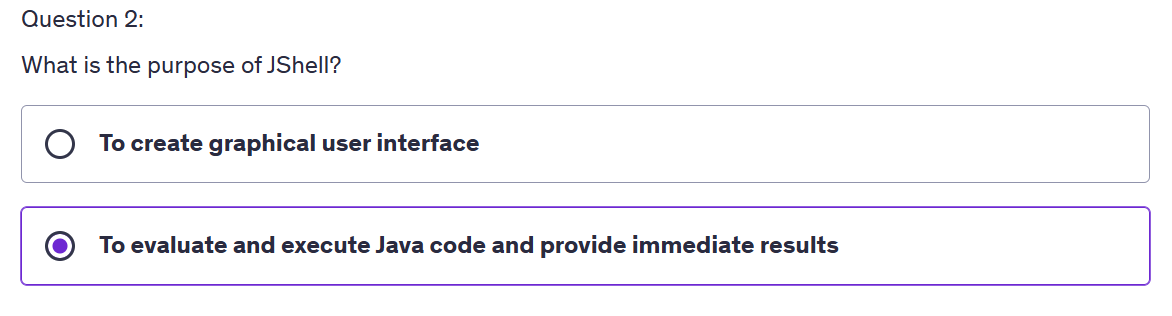
Relational before Logical,

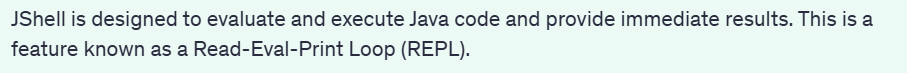
Assignment last.

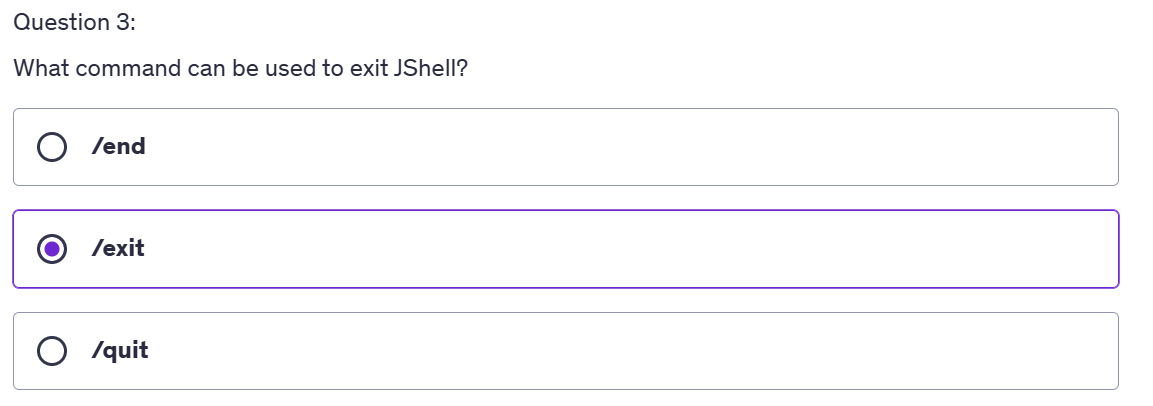
Quizzes:

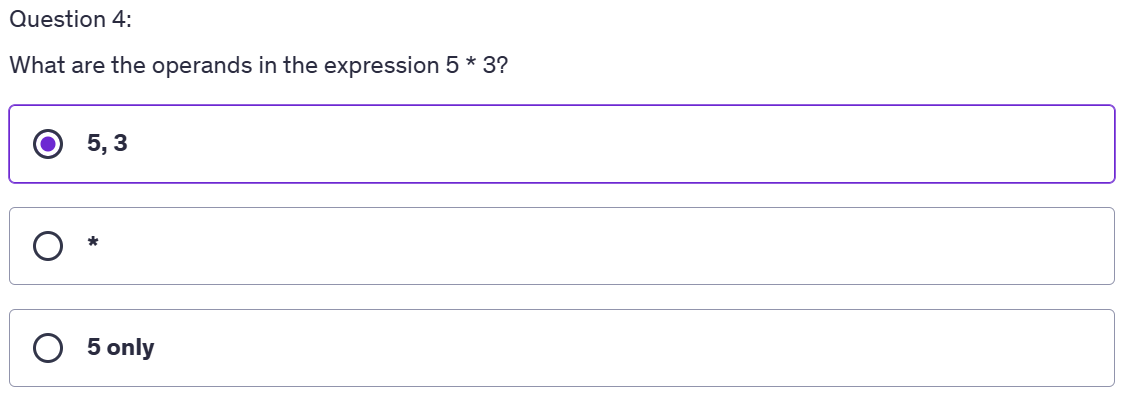


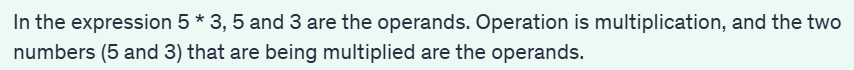


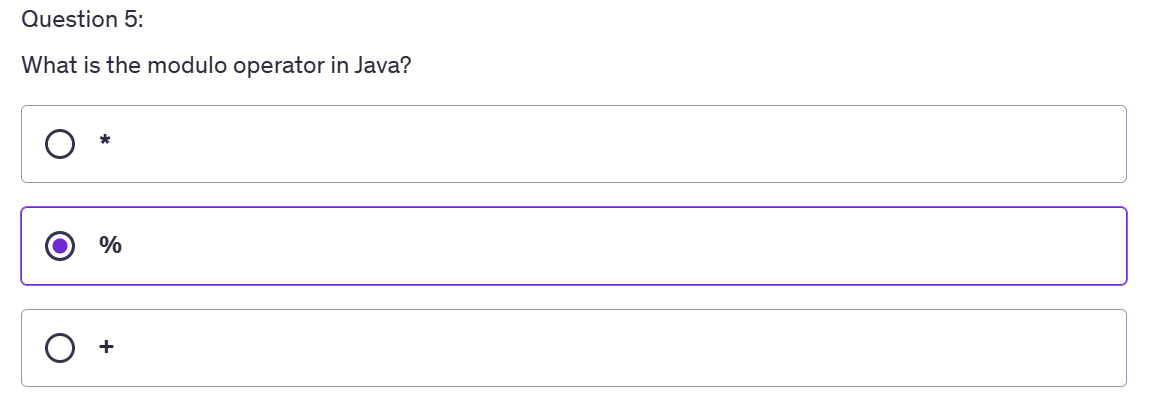


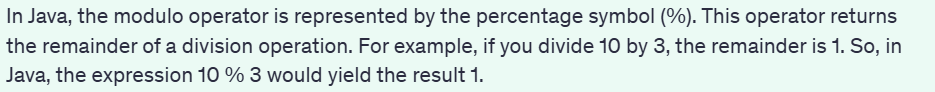


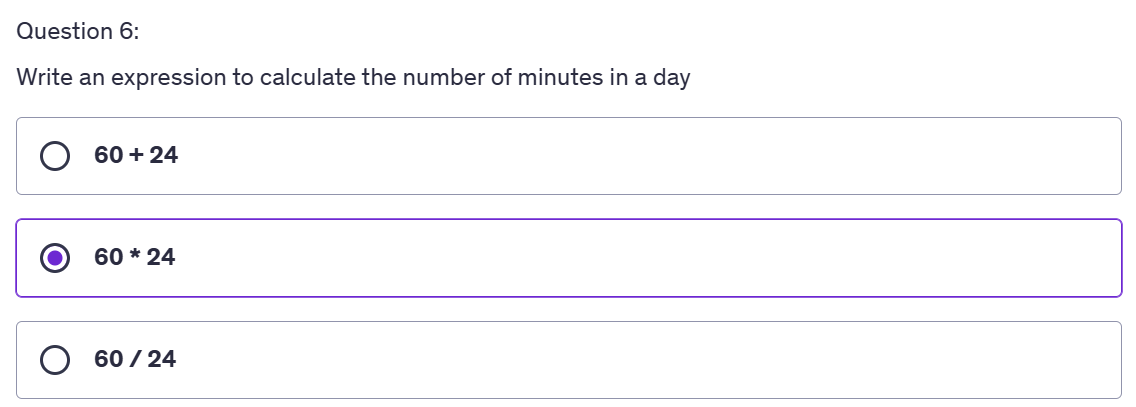


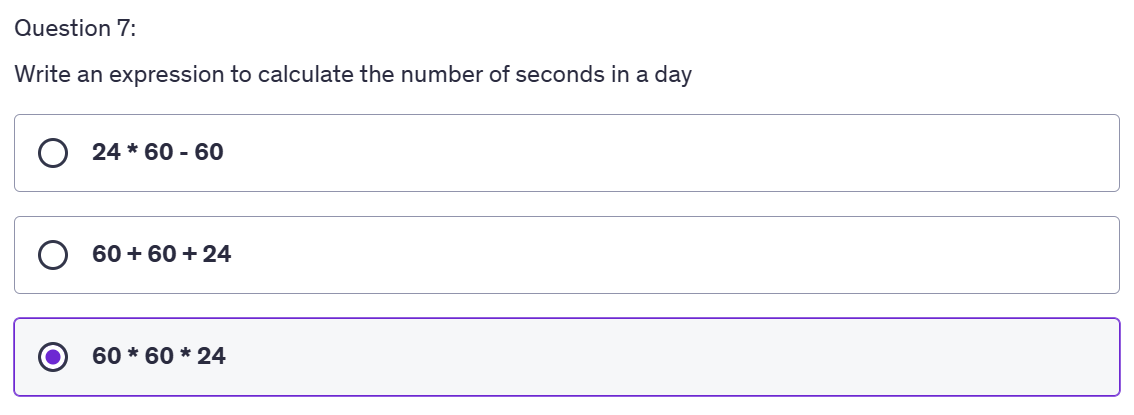


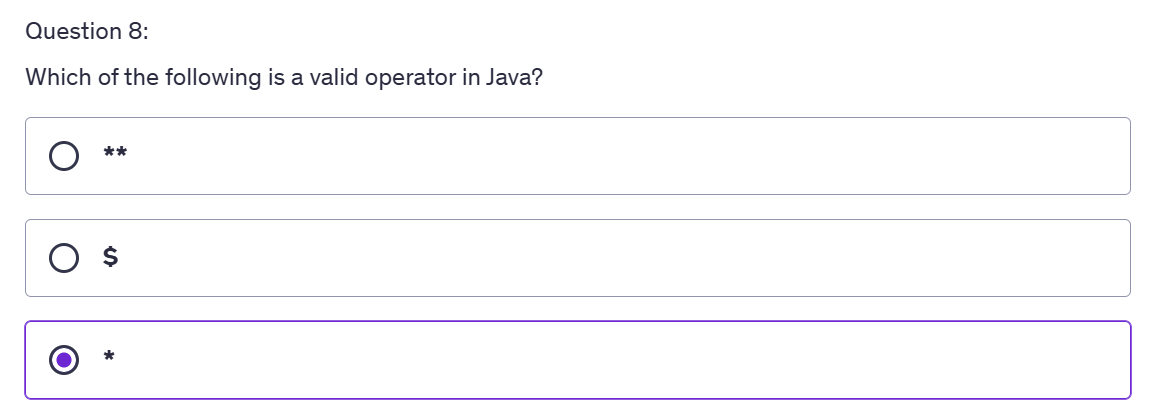


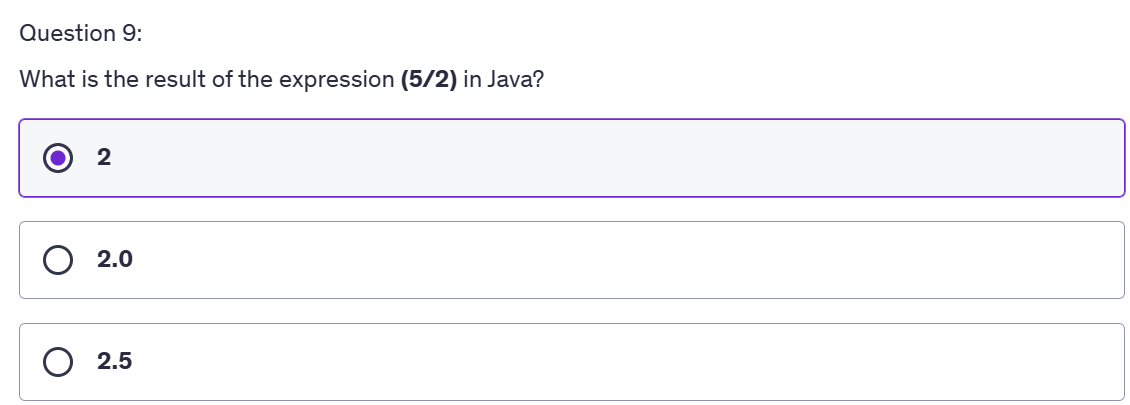


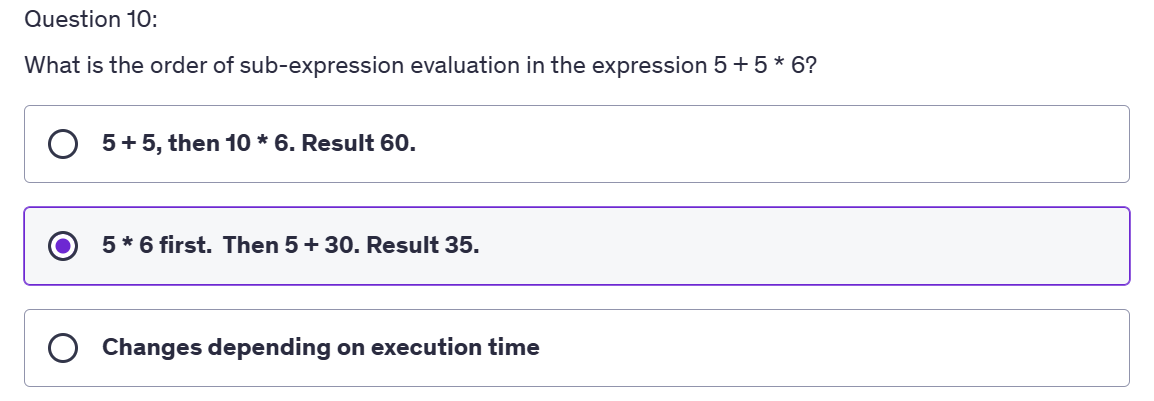












Method path(parameters)

**ESCAPE CHARACTERS:**

* In Java, an **escape character** is a **backslash \** followed by another character.
* It tells the compiler to interpret the sequence differently (not as plain text).
* Mainly used in **strings** and **characters**.

| **Escape Sequence** | **Meaning** | **Example** | **Output** |
| --- | --- | --- | --- |
| \' | Single quote | System.out.println('\''); | ' |
| \" | Double quote | System.out.println("\"Java\""); | "Java" |
| \\ | Backslash | System.out.println("\\"); | \ |
| \n | New line | System.out.println("Hello\nWorld"); | Hello World |
| \t | Tab (horizontal space) | System.out.println("Java\tRocks"); | Java Rocks |
| \r | Carriage return | System.out.println("Hello\rWorld"); | World (replaces start) |
| \b | Backspace | System.out.println("AB\bC"); | AC |
| \f | Form feed (page break, rarely used) | - | - |
| \uXXXX | Unicode character (4 hex digits) | System.out.println("\u0041"); | A |