Answer the following

1. Why there is need of loop in programming language.

Loops are a fundamental concept in programming and are crucial for various reasons. Here's why loops are needed:

1. Repetition of Tasks

Automation: Loops automate repetitive tasks, reducing the need for manually writing the same code multiple times. For example, processing each item in a list or performing a task a fixed number of times.

Efficiency: They help in executing a block of code multiple times efficiently without duplicating the code.

2. Dynamic Data Handling

Variable Data Size: Loops are essential for handling data structures with dynamic sizes, such as arrays, lists, or collections. They enable you to process each element of the data structure without knowing its size beforehand.

3. Simplification of Code

Readability: Using loops simplifies the code, making it more readable and maintainable. Instead of writing repetitive code, you use a loop to iterate over the data.

Reduction of Errors: By centralizing repetitive logic within a loop, you reduce the risk of errors that can occur with duplicated code.

4. Scalability

Handling Larger Inputs: Loops make your programs scalable. They allow your code to handle varying amounts of input data, making it more adaptable to different scenarios.

Flexibility: You can use loops to implement algorithms that require iterative steps, such as searching, sorting, or iterating over complex data structures.

5. Control Flow Management

Conditional Repetition: Loops provide mechanisms to repeat operations based on certain conditions. This is useful in scenarios where the number of iterations is not known in advance or depends on runtime conditions.

Examples of Loop Usage:

Iterating Through Collections: Accessing and manipulating elements in arrays or lists.

Repeated Calculations: Performing operations like summing numbers or calculating factorials.

User Input: Continuously prompting the user for input until valid data is provided.

Example Code

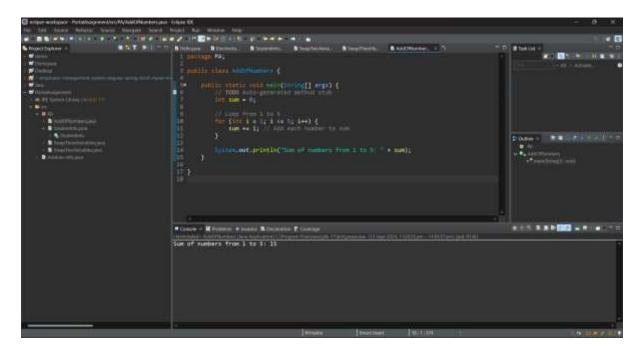
Here's a simple example using a for loop to calculate the sum of numbers from 1 to 5:

```
package PA;
public class AddOfNumbers {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
    int sum = 0;
```

```
// Loop from 1 to 5
    for (int i = 1; i <= 5; i++) {
        sum += i; // Add each number to sum
    }

    System.out.println("Sum of numbers from 1 to 5: " + sum);
}</pre>
```

Output:-



2. Write a difference between break and continue.

Aspect	break	continue
Purpose	Exits the current loop or	Skips the current iteration of a
	switch statement immediately.	loop and proceeds with the
		next iteration.
Effect on	Terminates the loop entirely.	Skips the remaining code in
Loop	Control moves to the	the current iteration and
	statement immediately	proceeds with the next
	following the loop.	iteration of the loop.
Use Case	Use when you need to exit the	Use when you want to skip the
	loop prematurely based on a	rest of the current loop
	specific condition.	iteration and move to the next
		iteration.

Impact on	Stops execution of the loop or	Does not exit the loop; only
Execution	switch block, possibly exiting	affects the current iteration.
	nested loops.	
Example	java while (condition)	java br>while (condition)
	{ if (someCondition)	{ str> if (someCondition)
	{ break; } //	{ continue; } //
	more code }	more code }

3. Differentiate between switch-case and else-if ladder.

Aspect	switch-case	else-if Ladder
Purpose	Evaluates a single expression	Executes code based on
	and branches based on its	multiple boolean
	value.	conditions.
Expression	Works with constants, typically	Works with any boolean
Type	integers, characters, or enums.	expression or condition.
Syntax	java switch (expression)	java if (condition1)
	{ > case value1: //	{ // code } else if
	code break; case	(condition2) { //
	value2: // code	code } else { //
	break; default: //	code }
	code }	
Readability	Generally more readable when	Can become less readable
	dealing with multiple discrete	with many else-if
	values.	conditions.
Performance	Often optimized by the	May require multiple
	compiler to use a jump table for	evaluations of conditions,
	faster execution.	especially if many else-if
		statements are present.
Use Case	Best for checking a single	Suitable for complex
	variable against a fixed set of	conditions involving
	possible values.	multiple variables or
		expressions.
Default Case	default case is optional, but can	else block is used to handle
	be used to handle values not	all conditions not covered
	matched by any case.	by previous if-else checks.
Fall-Through	case labels can fall through to	No fall-through; each else-
Behavior	the next case unless a break is	if condition is mutually
	used.	exclusive.

4. Differentiate between while loop and do-while loop.

Aspect	while Loop	do-while Loop
Condition	The condition is checked	The condition is checked
Check	before the loop body	after the loop body
	executes.	executes.
Initial	The loop body may not	The loop body is
Execution	execute if the condition is	guaranteed to execute at
	false initially.	least once.
Usage	Suitable when the number	Suitable when the loop
	of iterations is not known	must execute at least once,
	and the loop might not	regardless of the condition.
	need to run at all.	
Syntax	java >while (condition)	java do { //
	{ // code }	code } while
		(condition);
Example	java int i =	java < br > int i = 0; < br > do
	0; while (i < 5) { br>	{
	System.out.println(i);	System.out.println(i);
	i++; }	i++; } while (i <
		5);
Behavior	If the condition is false	The loop body will
	from the beginning, the	execute at least once, even
	loop body will not execute	if the condition is false
	even once.	initially.
Control	The loop may be skipped	The loop is executed at
Flow	entirely if the condition is	least once before checking
	false.	the condition.