Class 9

Clapter 10

Nested Loop

A. Tick (✓) the correct answer — solved with explanation

Q1. In a nested `do-while` loop, what happens if the inner loop evaluates to `false` on the first check?
Options:
a. run time error
b. the outer loop executes once
c. the inner loop execute once
d. both b and c

Answer: d. both b and c

Explanation: In a `do-while` the body executes first before the condition is checked. So when you have a nested `do-while`, the outer body (which contains the inner `do-while`) will run once, and the inner `do-while` body will also run once — even if its condition is false at the first check. No runtime error occurs.

Demo (Java)

```
public class NestedDoWhileDemo {
  public static void main(String[] args) {
    int outer = 1;
    do {
      System.out.println("Outer body start");
      int inner = 1;
      do {
        System.out.println("  Inner body executed once");
        inner++;
      } while (inner <= 0); // condition is false at first check (inner=2 >
0) so inner runs once
      System.out.println("Outer body end");
      outer++;
    } while (outer <= 0); // outer condition also false at first check, outer
still ran once
  }
}
```

If you run it you will see both outer and inner bodies printed once.

Q2. While loops are nested. Which loop needs to end abruptly for the entire construct to shut down?
Options:
a. the first outer loop
b. the middle loop
c. the innermost loop
d. none of these

Answer: a. the first outer loop

Explanation: In nested `while` loops the outermost loop controls the whole construct. If only the innermost or a middle loop ends, control returns to the containing loop which may continue. To stop the whole nested construct (i.e., to stop all iterations), the outermost loop must finish (or be terminated). An abrupt end (e.g., via a `break` labeled to the outer loop or a condition that becomes false) must target the outer loop to shut down the entire nested structure.

Example: Suppose you have `while(outer) { while(middle) { while(inner) { ... } } }` — even if `inner` finishes, `middle` or `outer` can still continue unless `outer` ends.

Q3. What is the output of the following code?

```
class NestedFor {
  public static void main(String s[]) {
    int sum = 56;
    for(int i = 3; i <= 6; i++) {
      for(int j = 8; j <= 10; j++) {
        sum += (i * j);
      }
    }
    System.out.println("sum = " + sum);
  }
}
```

Answer: `sum = 542`
Explanation: For each `i` (3..6), inner loop j runs 8..10. Sum of j for one i = 8+9+10 = 27. Contribution per `i` = `i * 27`. Total added = 27*(3+4+5+6) = 27*18 = 486. Initial sum 56 → 56 + 486 = **542**. *(Note: the multiple-choice options on the page did not include 542 — so the correct printed result is 542.)*

Q4. Consider:

```
int k = 0;
int m = 5;
int n = m;
while (k < n) {
  k++;
```

```
  n--;
}
System.out.println(k + n);
```

Answer: 5
Explanation: Iterations:

- start: k=0, n=5 → after 1st iter: k=1, n=4 (1<4 true)
- after 2nd iter: k=2, n=3 (2<3 true)
- after 3rd iter: k=3, n=2 (condition fails) → exit
  Print `k + n = 3 + 2 = 5`.

Q5. A `break` statement inside a loop like `while`, `for`, `do-while` causes the program execution ___ the loop.
Answer: Exit (it causes immediate exit from that loop).

B. Fill in the blanks (answers)

1. A loop within a loop is called a **nested loop**.
2. **break** statement is used to terminate a loop.
3. Continue statement **skips the remaining statements in the current iteration and continues with the next iteration**. (Short form: **skips** the current iteration.)
4. **A do-while** loop within a do-while loop is known as **nested do-while** loop.

C. Short answer type questions

1a. Differentiate between Nested `for` and Nested `do-while`.

- **Nested `for`**: both outer and inner loops are `for` loops. You know the loop-control (init; condition; update) in one line. Good when number of iterations known beforehand.
- `for(int i=1;i<=3;i++){`
- `  for(int j=1;j<=2;j++){`
- `    // body`
- `  }`
- `}`
- **Nested `do-while`**: inner and/or outer loops are `do-while`. Body executes at least once before condition is checked.
- `do {`
- `  do {`
- `    // body`
- `  } while(innerCondition);`
- `} while(outerCondition);`

1b. Break and Continue — difference

- `break`: immediately exits the innermost loop (or labeled loop if labeled). Execution continues after that loop.
- `continue`: skips the rest of current iteration and proceeds with next iteration of that loop.

2. Define Nested Loop.
A nested loop is a loop inside another loop. The inner loop runs completely for each single iteration of the outer loop.

3. Syntax of a nested `while` loop (example):

```
while(conditionOuter) {
  // outer body
  while(conditionInner) {
    // inner body
    // update inner control
  }
  // update outer control
}
```

4. Program to demonstrate use of `break` in a nested `while` loop (Java):

```
public class BreakInNestedWhile {
  public static void main(String[] args) {
    int i = 1;
    while (i <= 3) {
      int j = 1;
      while (j <= 4) {
        if (j == 3) {
          System.out.println("break at inner j=" + j);
          break; // exits inner loop only
        }
        System.out.println("i=" + i + ", j=" + j);
        j++;
      }
      i++;
    }
  }
}
```

This program shows `break` exiting only the inner loop; outer loop continues.

D. More unsolved programs — solutions / Java programs

Below are simple, clear Java programs for each required task. You can copy them directly.

---

1. Write a program to input n numbers and print all the prime numbers among them.

```
import java.util.*;
public class PrintPrimesFromList {
  public static boolean isPrime(int x) {
    if (x <= 1) return false;
    if (x <= 3) return true;
    if (x % 2 == 0) return false;
    for (int i = 3; i * i <= x; i += 2) {
      if (x % i == 0) return false;
    }
    return true;
  }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("How many numbers? ");
    int n = sc.nextInt();
    System.out.println("Enter " + n + " numbers:");
    for (int i = 0; i < n; i++) {
      int val = sc.nextInt();
      if (isPrime(val)) System.out.println(val + " is prime");
    }
    sc.close();
  }
}
```

---

2. Print all Armstrong numbers between 200 and 1000.
(Armstrong number for 3-digit: sum of cubes of digits = number)

```
public class Armstrong200to1000 {
  public static boolean isArmstrong(int x) {
    int t = x, sum = 0;
    while (t > 0) {
      int d = t % 10;
      sum += d * d * d;
      t /= 10;
    }
    return sum == x;
  }
  public static void main(String[] args) {
    for (int i = 200; i <= 1000; i++) {
      if (isArmstrong(i)) System.out.println(i);
    }
  }
}
```

Expected outputs in range: `370, 371, 407` (and 153 <200 not included).

---

3. Print all palindromic numbers between m and n (m < n).

```
import java.util.*;
public class PalindromesBetween {
  public static boolean isPalindrome(int x) {
    int orig = x, rev = 0;
    while (x > 0) {
```

```
      rev = rev * 10 + (x % 10);
      x /= 10;
    }
    return orig == rev;
  }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter m and n: ");
    int m = sc.nextInt();
    int n = sc.nextInt();
    for (int i = m; i <= n; i++) {
      if (isPalindrome(i)) System.out.println(i);
    }
    sc.close();
  }
}
```

---

4. Write a program to input "n" numbers and print the sum of all the Niven (Harshad) numbers.
(Niven/Harshad: divisible by sum of its digits.)

```
import java.util.*;
public class SumOfNiven {
  public static int digitSum(int x) {
    int s = 0;
    while (x > 0) { s += x % 10; x /= 10; }
    return s;
  }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("How many numbers? ");
    int n = sc.nextInt();
    int total = 0;
    System.out.println("Enter numbers:");
    for (int i = 0; i < n; i++) {
      int val = sc.nextInt();
      if (val != 0 && val % digitSum(val) == 0) total += val;
    }
    System.out.println("Sum of Niven numbers = " + total);
    sc.close();
  }
}
```

---

5. Write a program to print the sum of the series
Interpretation used: `s = 1/2! + 2/4! + 3/6! + ... + 10/20!` (numerator 1..10, denominators factorials of even numbers 2,4,...,20).

```
public class SeriesFactorial {
  static double fact(int n) {
    double f = 1;
    for (int i = 1; i <= n; i++) f *= i;
    return f;
  }
  public static void main(String[] args) {
    double s = 0.0;
    int numerator = 1;
    for (int k = 2; k <= 20; k += 2) {
```

```
      s += (double)numerator / fact(k);
      numerator++;
    }
    System.out.println("Sum = " + s);
  }
}
```

---

6. Write a program to print this series:

`s = 1 + (1+3) + (1+3+5) + ... + (1+3+5+... + nth odd)`

We know sum of first r odd numbers = r^2. So the series sum = 1^2 + 2^2 + 3^2 + ... + n^2 = n(n+1)(2n+1)/6. Program:

```
import java.util.*;
public class OddSeriesSum {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter n: ");
    int n = sc.nextInt();
    // Using formula for sum of squares:
    int sum = n * (n + 1) * (2 * n + 1) / 6;
    System.out.println("Sum = " + sum);
    sc.close();
  }
}
```

---

7. Write programs to print the following patterns

I'll give short Java code examples for typical pattern types (a — h). You can adapt `rows` or values as needed.

(a) Increasing numbers in rows (example shown on page)

```
public class PatternA {
  public static void main(String[] args) {
    int num = 1;
    for (int r = 1; r <= 4; r++) {
      for (int c = 1; c <= r; c++) {
        System.out.print(num + " ");
        num++;
      }
      System.out.println();
    }
  }
}
```

(b) Repeated digits pattern (example like page)

```
public class PatternB {
  public static void main(String[] args) {
    for (int r = 1; r <= 5; r++) {
      for (int c = 1; c <= r; c++) {
        System.out.print(r); // prints row number repeated
      }
      System.out.println();
    }
  }
}
```

(c) Decreasing numbers in each row (e.g. `10 9 8 7`)

```
public class PatternC {
  public static void main(String[] args) {
    int start = 10;
    for (int r = 0; r < 4; r++) {
      for (int c = 0; c < 4 - r; c++) {
        System.out.print((start - c) + " ");
      }
      System.out.println();
    }
  }
}
```

(d) Reverse counting per row (example `1`, `2 1`, `3 2 1`, ...)

```
public class PatternD {
  public static void main(String[] args) {
    for (int r = 1; r <= 4; r++) {
      for (int c = r; c >= 1; c--) {
        System.out.print(c + " ");
      }
      System.out.println();
    }
  }
}
```

(e) Repeated digit rows (1, 2 2, 3 3 3 ...)

```
public class PatternE {
  public static void main(String[] args) {
    for (int r = 1; r <= 5; r++) {
      for (int c = 1; c <= r; c++) {
        System.out.print(r + " ");
      }
      System.out.println();
    }
  }
}
```

(f) A block of `1`s repeated (e.g., 5 rows, each with 5 ones)

```
public class PatternF {
  public static void main(String[] args) {
    for (int r = 0; r < 5; r++) {
      for (int c = 0; c < 5; c++) System.out.print("1");
      System.out.println();
    }
  }
}
```

(g) Decreasing number of `x` per row:

```
public class PatternG {
  public static void main(String[] args) {
    int rows = 4;
    for (int r = rows; r >= 1; r--) {
      for (int c = 1; c <= r; c++) System.out.print("x");
```

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

(h) Alphabet blocks (A, B C, D E F, ...)

```
public class PatternH {
  public static void main(String[] args) {
    char ch = 'A';
    for (int r = 1; r <= 4; r++) {
      for (int c = 1; c <= r; c++) {
        System.out.print(ch + " ");
        ch++;
      }
      System.out.println();
    }
  }
}
```