SOLUTIONS & SCHEME IAT2
Date 5th Aug 2022

Course Name: Object oriented programming concepts Course Code: 18CS45

Q.No.	Solution Scheme	Marks
1a.	Describe Different Types of Operators in java	6M

Ans

Operators are symbols that perform special operations on one, two or three operands and then return a result.

In Java, operators are divided into four groups
1. Arithmetic
2. Bitwise
3. Relational
4. Logical
5. Short-Circuit Logical Operators
6.? Question mark Operator

1. Arithmetic Operators:

- Arithmetic operators are used in mathematical expressions.
- The operands of the arithmetic operators must be of numeric type.
- It can't be used on boolean type.
- It can be used on char type as char type in Java is a subset of int.
- The various arithmetic operators are shown in the table below

Sl. No.	Operator	Result
1	+	Addition
2	-	Subtraction (Also unary minus)
3	*	Multiplication
4	/	Division
5	%	Modulus
6	++	Increment
7	+=	Addition Assignment
8	-=	Subtraction Assignment
9	*=	Multiplication Assignment
10	/=	Division Assignment
11	%=	Modulus Assignment
12	--	Decrement

The Basic Arithmetic Operators:

- The basic arithmetic operations are -

1. addition
2. subtraction
3. multiplication
4. division

- The minus operator also has a unary form that negates its single operand.
- When the division operator is applied to an integer type, there will be no fractional component attached to the result.
- The following program demonstrates the arithmetic operations -

```
// Program to demonstrate the basic arithmetic operators
class BasicMath {
        public static void main( String args[ ]) {
                System.out.println("Integer Arithmetic");
                int a = 1 + 1;
                int b = a * 3;
                int c = b / 4;
                int d = c – a;
                int e = -d;
System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                System.out.println("d = " + d);
                System.out.println("e = " + e);

                System.out.println("Floating Point Arithmetic");
                double da = 1 + 1;
                double db= da * 3;
                double dc = db / 4;
                double dd = dc - da;
                double de = -dd;

                System.out.println("da = " + da);
                System.out.println("db = " + db);
                System.out.println("dc = " + dc);
                System.out.println("dd = " + dd);
                System.out.println("de = " + de);
        }
}
```

Output:

```
Integer Arithmetic
a = 2
b = 6
c = 1
d = -1
e = 1
Floating point arithmetic
da = 2.0
db = 6.0
dc = 1.5
dd = -0.5
de = 0.5
```

The modulus operator:

- The modulus operator, %, returns the remainder of a division operation.
- It can be applied to floating-point types as well as integer types.
- The following program demonstrates the % operator

```
// Demo of  % operator
```

```
class Modulus {
        public static void main(String args[ ]) {
                int x = 42;
double y = 42.25;

                System.out.println("x mod 10 = " + x % 10);
                System.out.println("y mod 10 = " + y % 10);
        }
}
```

Output:

x mod 10 = 2
y mod 10 = 2.25

Arithmetic Compound Assignment Operators:

- Compound assignment operators are special operators that are used to combine an arithmetic operation with an assignment operation.
- Statement like the following
  a = a + 4;
  can be rewritten as
  a += 4;
- The above statement uses the += compound assignment operator. Both statements perform the same action. They increase the value of a by 4.
- There are compound assignment operators for all arithmetic, binary operators.
- Any statement of the form
  var = var op expression;
  can be rewritten s
  var op= expression;
- **Advantages:**
  1. They save a bit of typing because they are "shorthand" for their equivalent long forms.
  2. They are implemented more efficiently by the Java run-time system than their equivalent long forms.
- Hence professionally written Java programs use compound assignment operators.
- The following program illustrates several op= assignments in action

// Demo program to illustrate compound assignment operators

```
class OpEquals {
        public static void main(String args [ ]) {
                int a = 1;
                int b = 2;
                int c = 3;

                a += 5;
                b *= 4;
                c += a * b;
                c %= 6;
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
        }
}
```

Output:

a = 6
b = 8
c = 3

Increment and Decrement operator:

- The ++ and – are Java's increment and decrement operators respectively.
- The increment operator increases its operand by one.
- The decrement operator decreases its operand by one.
- The statement
  x = x + 1;
  can be rewritten in Java using increment operator as
  x++;
- Similarly, the statement
  x = x – 1; is same as
  x--;
- Increment and decrement operators appear both in postfix form and prefix form.
- In postfix form, the operator follows the operands.
- In prefix form the operator precedes the operands.
- For statements like
  x++;
  --y;
  there is no difference between prefix and postfix forms.
- The prefix and postfix forms matter a lot when the increment and/or decrement operators are part of a larger expression.
- In the prefix form, the operand is incremented or decremented before the value is obtained for use in the expression.
- In the postfix form, the previous value is obtained for use in the expression and then the operand is modified.
- Prefix example, Consider the statements -
  x = 42;
  y = ++x;
  Here, the increment occurs before x is assigned to y. So y = 43 and x = 43. Thus, the above line is equivalent to the following two statements
  x = x + 1;
  y = x;
- Postfix example: Consider the statements -
  x = 42;
  y = x++;
  Here, the value of x is obtained before the increment operator is executed. So y = 42 and x = 43. Thus, the above line is equivalent to the following two statements -
  y = x;
  x = x + 1;
- The following program demonstrates the increment operator

```
// class IncDec {
public static void main (String args [ ])  {
        int a = 1;
        int b = 2;
        int c;
        int d;
        c = ++b;
        c = a++;
        c++;
```

```
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                System.out.println("d = " + d);
        }
}
```

Output:
a = 2
b = 3
c = 4
d = 1

Relational Operators:

- The relational operators determine the relationship between the two operands.
- They determine equality and ordering.
- The relational operators are

Sl. No.	Operator	Result
1	==	Equal to
2	!=	Not Equal to
3	>	Greater than
4	<	Less than
5	>=	Greater than or equal to
6	<=	Less than or equal to

- The outcome of these operations is a boolean value.
- The relational operators are used in the expressions that control the if statement and various loop statements.
- Any type in Java, including integers, floating-point numbers, characters and Booleans can be compared using the equality test, ==, and the inequality test, !=.
- only numeric types can be compared using the ordering operators. That is, only integer, floating-point, and character operands may be compared to see which is greater or less than the other.
- Example:

```
int a = 4;
int b = 1;
boolean c = a < b;
```

- The result of a < b (which is false) is stored in c.
- The C/C++ statements

```
int done;
        if (!done)    ...
        if (done)   ...
        must be written like this
        if ( done == 0)  ...
        if ( done != 0)  ...
```

- Java does not define true and false in the same way as C/C++. In C/C++, true is any nonzero value and false is zero.
- In Java, true and false are non numeric values that do not relate to zero or nonzero. Therefore to test for zero and non-zero, we must explicitly employ one or more of the relational operators.

ogical Operators:

- The boolean logical operators operate only on boolean operands.
- All of the binary logical operators combine two boolean values to form a resultant boolean value.

Sl. No.	Operator	Result
1	&	Logical AND
2	|	Logical OR
3	^	Logical XOR (Exclusive OR)
4	||	Short-circuit OR
5	&&	Short-circuit AND
6	!	Logical unary NOT
7	&=	AND assignment
8	|=	OR assignment
9	^=	XOR assignment
10	==	Equal to
11	!=	Not equal to
12	?:	Ternary if-then-else

The following table shows the effect of each logical operation:

A	B	A | B	A & B	A ^ B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

```
// Program to demonstrate the boolean logical operators
class BoolLogic {
        public static void main(String args[ ]) {
                boolean a = true;
                boolean b = false;
                boolean c = a |  b;
                boolean d = a & b;
                boolean e = a ^ b;
                boolean f = (!a & b) | (a & !b);
                boolean g = !a;

                System.out.println(“              a = “ + a);
                System.out.println(“              b = “ + b);
                System.out.println(“          a | b = “ + c);
```

```
                System.out.println("        a & b = " + d);
                System.out.println("        a ^ b = " + e);
                System.out.println("!a&b | a& !b= " + f);
        }
}
```

Output:

```
          a = true
          b = false
       a | b = true
      a & b = false
        a^b = true
                a&b | a&!b = true
!a = false
```

Short-Circuit Logical Operators:

- Java provides two Boolean operators not found in many other computer languages. These are secondary versions of the Boolean AND and OR operators and are known as short-circuit operators.
- From the table we can see that, the OR operator results in true when A is true, no matter what B is. Similarly, the AND operator results in false when A is false, no matter what B is.
- **When we use || and && forms, rather than | and & forms of these operators, Java will not bother to evaluate the right-hand operand when the outcome of the expression can be determined by the left operand alone.**
- Example
  if (denom !=0 && num /denom > 10)
  As the short circuit form of && is used, there is no risk of causing a run-time exception when denom is zero.
- If this line of code were written using the single & version of AND, both sides would be evaluated, causing a run-time exception when denom is zero.
- Its a standard practice to use the short-circuit forms of AND and OR in cases involving Boolean logic, leaving the single character versions exclusively for bitwise operations.

he ? Operator:

- Java includes a ternary (three-way) operator that can replace certain types of if-then-else-statements.
- The operator is ?.
- General form:
  expression1 ? Expression2 : expression3
- expression1 can be any expression that evaluates to a boolean value.
- If expression1 is true then expression2 is evaluated, else, expression3 is evaluated.
- Example:

```
ratio = denom == 0 ? 0 : num /denom;
```

- if denom equals zero, then the expression between the question mark and the colon is evaluated and used as the value of the entire ? Expression.
- If denom is not equal to zero, then the expression after the colon is evaluated and used for the value of the entire ? Expression.

	• The result is then assigned to ratio.
	• Example program
	ass Ternary {
	public static void main(String args[ ]) {
	int i, k;
	i = 10;
	k = i < 0 ? -i : i;
	System.out.println(“Absolute value of “ + i + “ is “ + k);
	
	i = -10;
	k = i < 0 ? -i : i;
	System.out.println(“Absolute value of “ + i + “ is “ + k);
	}
	}
	Output:
	Absolute value of 10 is 10
	Absolute value of -10 is 10	

1b	Guess the output and justify your answer.
	class Operator
	{
	public static void main(String[] args)
	{
	int a = 30;
	int b = 20;
	int c = a++ + ++b;
	int d = --a + --b + c--;
	int e = a +b +c + d--;
	int f = -a + b-- + -c - d++;
	int sum = a + b + c + d + e + f;
	System.out.println("sum = " + sum);
	}
	}
		4M

Ans	sum = 241 a= 30 b= 19 c= 50 d= 101 e= 201 f= -160 sum = 241	4M

2 a	List all and explain any 5 java buzzwords. .	6M
Ans	The following is the list of Java buzzwords 1. Simple 2. Secure 3. Portable 4. Object-Oriented 5. Robust 6. Multi threaded 7. Architecture-neutral 8. Interpreted 9. High performance 10. Distributed 11. Dynamic. [1M] 1. Simple: Java was designed to be easy for the professional programmer. For those who have already understood the basic concepts of object-oriented programming, and for an experienced C++ programmer learning Java will be even easier as Java inherits the C/C++ syntax and many of the object-oriented features of C++. 2. Secure: Java provides security. The security is achieved by confining an applet to the Java execution environment and not allowing it access to other parts of the computer. The ability to download applets with confidence that no harm will be done and that no security will be breached is considered by many to be the single most innovative aspect of Java. 3. Portable: Portability is a major aspect of the Internet because there are many different types of computers and operating systems connected to it. If a Java program were to be run on virtually any computer connected to the Internet, there needs to be some way to enable the program to execute on different systems. Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments because only the JVM needs to be implemented for each platform. Once the run-time package exists for a given system, any Java program can run on it. 4. Object-Oriented: Java has a clean, usable, pragmatic approach to objects. The object model in Java is simple and easy to extend. The primitive types, such as integers, are kept as high-performance non-objects. 5. Robust: The ability to create robust programs was given a high priority in the design of Java. To gain reliability, Java restricts programmer in a few key	

areas to force the programmer to find mistakes early in program development. Also, Java frees the programmer from having to worry about many of the most common causes of programming errors. As Java is a strictly typed language, it checks code not only at run time but also during compilation time. As a result, many hard-to-track-down bugs that often turn up in hard-to-reproduce run-time situations are simply impossible to create in Java.
The two features – Garbage collection and Exception handling enhance the robustness of Java Programs.

a) Garbage Collection:
In C/C++, the programmer must manually allocate and free all dynamic memory which sometimes leads to problems, because programmers will either forget to free memory that has been previously allocated or, try to free some memory that another part of their code is still using. Java eliminates these problems by managing memory allocation and deallocation. De-allocation is completely automatic because Java provides garbage collection for unused objects.

b) Exception Handling:
Exceptional conditions in traditional environments arise in situations such as “division by zero” or “file not found” which are managed by clumsy and hard-to-read constructs. Java helps in this area by providing object oriented exception handling.

6. Multithreaded Programming:
Multi threaded Java supports multithreaded programming, which allows the programmer to write programs that do many things simultaneously. Java provides an elegant solution for multi process synchronization that enables the programmer to construct smoothly running interactive systems. Java’s easy-to-use approach to multithreading allows the programmer to think about the specific behavior of the program rather than the multitasking subsystem.
7. Architecture-neutral
The main issue for the Java designers was that of code longevity and portability. One of the main concerns of programmers is that there is no guarantee that their program will run tomorrow even on the same system. Operating system upgrades, processor upgrades and changes in core system resources together make a program malfunction. Java is been designed with the goal “write once and run anywhere, anytime, forever”, and to a great extent this goal is accomplished.

8. Interpreted and High Performance:
Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. This code can be executed on any system that implements the Java Virtual Machine. Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using just-in-time compiler.

9. Distributed:
As C is to system programming, Java is to Internet programming. Java is designed for the distributed environment of the Internet because it handles TCP/IP protocols. Accessing a resource using a URL is not much different

	from accessing a file. Java also supports Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network. 10. Dynamic: Java programs carry with them a substantial amount of run-time type information that is used to verify and resolve accesses to objects at run-time. This makes it possible to dynamically link code in a safe manner. Small fragments of bytecode may be dynamically updated on a running system. Explanation of any five will carry 5Marks	
2b	Explain Type Conversion and Casting in Java	4M

Answer	Java's Automatic Conversions:
	When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:
	◦ The two types are compatible.
	◦ The destination type is larger than the source type
	
	When these two conditions are met, a widening conversion takes place. For widening conversions, the numeric types, including integer and floating-point types are compatible with each other. There are no automatic conversions from the numeric types to char or boolean. Also char or boolean are not compatible with each other. Java also performs an automatic type conversion when storing a literal integer constant into variables of type byte, short, long or char.
	Casting Incompatible Types:
	If we want to assign an int value to a byte variable, conversion will not be performed automatically, because a byte is smaller than an int. This kind of conversion is called narrowing conversion since we are explicitly making the value narrower so that it will fit into the target type. To create a conversion between the two incompatible types, we must use a cast. A cast is simply an explicit type conversion. The general form of cast is - (target-type) value
	target-type specifies the desired type to convert the specified value to. For example to cast an int to a byte
	
	int a=20;
	byte b;
	b= (byte) a;
	If the integer value is larger than the range of a byte, it will be reduced modulo (the remainder of an integer division by the) byte's range. A different type of conversion called truncation will occur when a floating-point value is assigned to an integer type. Integers do not have fractional components. Hence when a floating point value is assigned to an integer type, the fractional component is lost. For example, if the value 1.23 is assigned to an integer, the resulting value will be 1. The 0.23 will be truncated. If the size of the whole number component is too large to fit into the target integer type, then the value will be reduced
	modulo the target type's range.	

<table>
<tr><td>3a</td><td>What will be the out put for fallowing statements.
Assume a=30 , b=20, x= -1
1. a&b
2. a|b
3.a^b
4 b>>3
5.a<<2
6.x>>>24</td><td>6M</td></tr>
<tr><td>Ans</td><td>1. a&b =20
2. a|b=30
3.a^b=10
4 b>>3=2
5.a<<2=120
6.x>>>2=1073741818</td><td></td></tr>
<tr><td>3b</td><td>Discuss Access Specifiers in java</td><td>4</td></tr>
<tr><td>Ans</td><td>

The three access specifiers, private, public, and protected, provide a variety of ways to produce the many levels of access required by these categories.
While Java’s access control mechanism may seem complicated, we can simplify it as follows. Anything declared public can be accessed from anywhere. Anything declared private cannot be seen outside of its class.
When a member does not have an explicit access specification, it is visible to subclasses as well as to other classes in the same package.
This is the default access. If you want to allow an element to be seen outside your current package, but only to classes that subclass your class directly, then declare that element protected.
A non-nested class has only two possible access levels: default and public.
When a class is declared as public, it is accessible by any other code.
If a class has default access, then it can only be accessed by other code within its same package.
When a class is public, it must be the only public class declared in the file, and the file must have the same name as the class.

<table>
<tr><th></th><th>Private</th><th>No Modifier</th><th>Protected</th><th>Public</th></tr>
<tr><td>Same class</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
<tr><td>Same package subclass</td><td>No</td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
<tr><td>Same package non-subclass</td><td>No</td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
<tr><td>Different package subclass</td><td>No</td><td>No</td><td>Yes</td><td>Yes</td></tr>
<tr><td>Different package non-subclass</td><td>No</td><td>No</td><td>No</td><td>Yes</td></tr>
</table>
</td><td></td></tr>
</table>

4	Differentiate between for and foreach statements in java, Write a java program to sum only first 5 elements of an array {1,2,3,4,5,6,7,8,9,10} using foreach loop	4M

Ans	For loop
	
	1. Here in for loop we can change counter as per our wish.
	2. can replace elements at any specific index.
	3. can iterate in both increment and decrement order.
	for each loop
	
	1. Executes in a sequential manner. Counter will increase by one.
	2. can't replace element at specific index since there is no access to index.
	3. we can only iterate in incremental order cannot decrement.
	
	A java program to sum only first5 elements of an array {1,2,3,4,5,6,7,8,9,10}using foreach loop,
	
	`class ForEach2 {`
	`  public static void main(String args[]) {`
	`  int sum = 0;`
	`  int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };`
	`  // use for to display and sum the values`
	`  for(int x : nums) {`
	`  System.out.println("Value is: " + x);`
	`  sum += x;`
	`  if(x == 5) break; // stop the loop when 5 is obtained`
	
	`}`
	
	`  System.out.println("Summation of first 5 elements: " + sum);`
	`  }`
	
	`}`
	
	This is the output produced:
	
	Value is: 1
	Value is: 2
	Value is: 3
	Value is: 4
	Value is: 5
	Summation of first 5 elements: 15	

5	Create a Java class called Student with USN, Name, Branch, Phone as variables within it. Write a Java program to create n student objects and print the USN, Name, Branch and Phone of these objects with suitable heading...	6M

Ans	import java.util.*;
	
	class Student1
	{
	public String usn, name, branch, phone;
	
	public Student1(String u, String n, String b, String ph)
	{
	usn=u;
	name=n;
	branch=b;
	
	phone=ph;
	}
	}
	
	class Lab1A
	{
	public static void main(String[] args)
	{
	
	Scanner in=new Scanner(System.in); // create the object of //scanner
	
	System.out.println("enter the total number of students:-");
	
	int n=in.nextInt(); // user input 2
	
	Student1[] st=new Student1[n]; // declaring the array
	
	String usn, name, branch, phone;
	
	for(int i=0;i<n;i++)
	{
	System.out.println("\n enter details of student:"+(i+1));
	
	System.out.println("USN:-");
	usn=in.next(); //1,2
	
	System.out.println("Name:-");
	name=in.next();// abc,efg
	
	System.out.println("Branch:-");
	branch=in.next();// ise,ise
	
	System.out.println("Phone number:-");
	phone=in.next(); //1234,4567
	
	st[i]=new Student1(usn, name, branch, phone);
	}	

```
System.out.println("Student details");// heading

System.out.println("USN\tName\tBranch\t\tPhone\t"); // heading

for(int i=0;i<n;i++)
{
System.out.println(st[i].usn+"\t"+st[i].name+"\t"+st[i].branch+"\t\t"+st[i].phone);
}
}
}
```

```
OUTPUT :

Enter total number of students:-
2

Enter details of student 1
USN:1CR15IS011
Name:Ram
Branch:IS
Phone number:9987867876

Enter details of student 2
USN:1CR15IS055
Name:Sham
Branch:IS
Phone number:8897876567

Student Details
USN         Name        Branch      Phone
1CR15IS011 Ram         CS          9987867876
1CR15IS055 Sham        CS          8897876567
```

6	Discuss the below terms with respect to Exception handling 1. Try 2. Catch 3. Throw 4. Throws 5.Finally. Also Write a Java program to read two integers a and b. Compute a/b and print, when b is not zero. Raise an exception when b is equal to zero.	4M
Ans	• Java exception handling is managed via five keywords: try, catch, throw, throws, and finally • Program statements that you want to monitor for exceptions are contained within a try block. • If an exception occurs within the try block, it is thrown. Your code can catch this exception (using catch) and handle it in some rational manner. • System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword throw. • Any exception that is thrown out of a method must be specified as such by a throws clause. • Any code that absolutely must be executed after a try block completes is put in a finally block.	

This is the general form of an exception-handling block:

```
try {
// block of code to monitor for errors

}

catch (ExceptionType1 exOb) {
// exception handler for ExceptionType1

}

catch (ExceptionType2 exOb) {
// exception handler for ExceptionType2

}
// ... finally {

// block of code to be executed after try block ends }
```

Program:

```
import java.util.*;
class MyException extends Exception {
public String toString() {
return &quot;Denominator is 0! Division by zero ERROR&quot;;
}
}

public class Main
{
public static void main(String[] args)
{
int a,b,quotient;
Scanner s = new Scanner(System.in);
System.out.println(&quot;Enter Numerator:&quot; );
a = s.nextInt();
System.out.println(&quot;Enter Denominator:&quot; );
b = s.nextInt();
try
{
if (b==0)
throw new MyException();
else {
quotient=a/b;
System.out.println(&quot;Quotient=&quot; + quotient);
```

```
}

}
catch(MyException ae)
{
System.out.println(ae);
}
}
}
```

OUTPUT 1:

```
Enter a :-2
Enter b:-5
Result a/b=0.4
```

OUTPUT 2:

```
Enter a :-2
Enter b:-0
Error !!!!: Denominator is 0! Division by zero ERROR
```