USN | | | | | | | | | | |

Internal Assessment Test 1 – December 2023

Sub:	Object Oriented Programming with JAVA					Sub Code:	BCS306A	Branch:	AIML & CSE(AIML)
Date:	/12/23	Duration:	90 minutes	Max Marks:	50	Sem/Sec:	III -A,B&C		OBE

		Answer any FIVE FULL Questions	MARKS	CO	RBT
1	a	Explain 3 oop principles	[6]	1	L1
	b	With a an example ,explain in working of Short circuit logical and >>>(unsigned right shift).	[4]		L1
2	a	Explain logical and Bitwise operators with an example (06M)	[6]	1	L3

	b	`publicclass Example {`
		`publicstaticvoid main(String[] args) {`
		`int a;`
		`for(a=0;a<3;a++) {`
		`int b=-1;`
		`System.out.println(""+b);`
		`b=50;`
		`System.out.println(""+b);`
		`}`
		`}`
		`}`
		What is the output of the above code? If you insert another 'int b' outside the for loop, what is the output.
			[4]	1	L2

3	a	what is typecasting? Illustrate with an example, what is meant by automatic type promotion	[6]	1	L2
	b	How to declare two dimensional arrays in java ? explain with an simple example.	[4]	1	L2
4	a	Explain switchcase with an example.	[6]	1	L2
	b	Differentiate between for loop and for each with an example.	[4]	1	L2
5	a	Discuss Lexical issues in JAVA program.	[6]	1	L2
	b	with an example , explain ternary operator.	[4]	1	L2
6	a	Explain class and objects with an example.	[5]	2	L2
	b	Write a java program to print even numbers from 0 to 100.	[5]	1	L2

Scheme

Question #	Description	Marks Distribution		Max Marks
1 a	Each OOP principal	2 M 2 M 2M	6M	6M
1 b	Logical Short circuit >>> Unsigned right shift	2 M 2 M	4M	4M
2a	Explanation of logical and Bitwise operators with an example (06M)	3+3	6M	6M

2b	`publicclass Example {`
	`publicstaticvoid main(String[] args) {`
	`int a;`
	`for(a=0;a<3;a++) {`
	`int b=-1;`
	`System.out.println(" "+b);`
	`b=50;`
	`System.out.println(" "+b);`
	`}`
	`}`
	`}`
	For the output of the above code
	Inserting another 'int b' outside the for loop, what is the output.
		2+2 M	4M	4M

3 a	Explanation of typecasting. Illustrating with an example, automatic type promotion+ Type promotion rules+ example	1+2+1+2	6M	6M
3b	Declaring two dimensional arrays in java explanation with an simple example.	2M 2M	4M	4M
4 a	a. Switch syntax b. Simple Switch , c. Nested Switch suitable example.	2 M 2 M 2M	6M	6M

4b	Differences between for loop and for each with an example.	4M	4M	4M
5 a	Lexical issues in JAVA program. whitespace, Identifiers, Literals, Comments, Separators, Keywords	1+1+1+1+1+1	6M	6M
5b	with an example , explanation of ternary operator.	2+2	4M	4M
6 a	Definition of class syntax. object with syntax.	1+2+2	5M	5M
6b	Write a java program to print even numbers from 0 to 100.	5M	5M	5M

Solution

Q1 a> Explain three OOP principles.

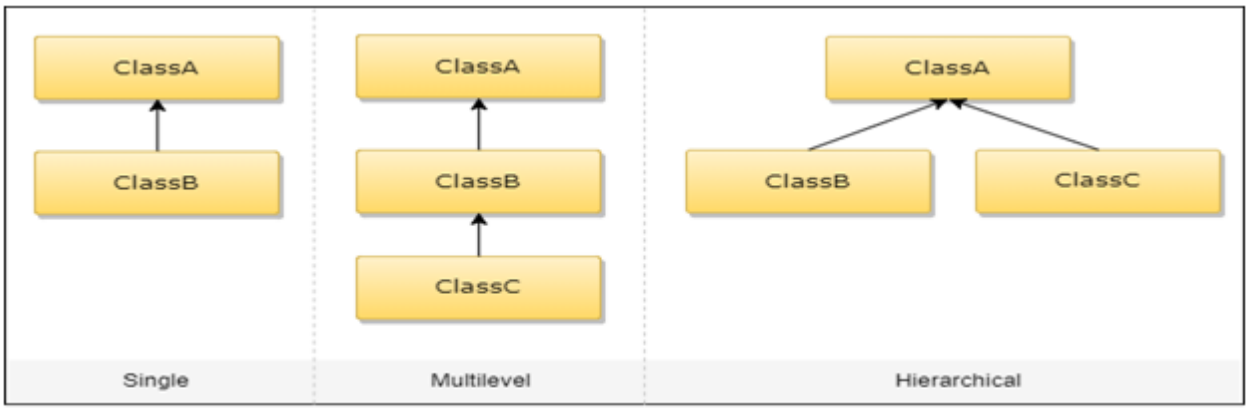
Encapsulation: Encapsulation can be defined as the procedure of casing up of codes and their associated data jointly into one single component.

In simple terms, encapsulation is a way of packaging data and methods together into one unit. Encapsulation gives us the ability to make variables of a class keep hidden from all other classes of that program or namespace.

Hence, this concept provides programmers to achieve data hiding. Programmers can have full control over what data storage and manipulation within the class

Inheritance: Inheritance can be defined as the procedure or mechanism of acquiring all the properties and behavior of one class to another, i.e., acquiring the properties and behavior of child class from the parent class.

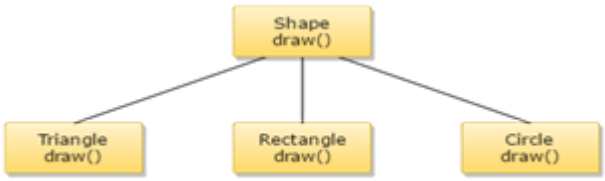
Java supports three types of inheritance. These are:



Java Inheritance

Polymorphism: The word polymorphism means having multiple forms. The term Polymorphism gets derived from the Greek word where poly + morphos where poly means many and morphos means forms.

- Static Polymorphism
- Dynamic Polymorphism.



Polymorphism

1b> use of short circuit property with example.

These are secondary versions of the Boolean AND and OR operators, and are known as short-circuit logical operators. As you can see from the preceding table, 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.

a=10 b=0

if(b&&a) here since b is 0; result is going to be 0; no need to check a

if(a||b) here since a is non zero; result is going to be 1; no need to check b

unsigned Right shift(>>>)

The Unsigned Right Shift/Shift Right Zero Fill: Always shifts zeros into the high-order bit.

For example, if you are shifting something that does not represent a numeric value, you may not want sign extension to take place. *This situation is common when you are working with pixel-based values and graphics.* In these cases, you will generally want to shift a zero into the high-order bit no matter what its initial value was.

value >>> num

The following code fragment demonstrates the >>>. Here, a is set to –1, which sets all 32 bits to 1 in binary. This value is then shifted right 24 bits, filling the top 24 bits with zeros, ignoring normal sign extension. This sets a to 255.

int a = -1;

a = a >>> 24;

Here is the same operation in binary form to further illustrate what is happening:

11111111 11111111 11111111 11111111 –1 in binary as an int

>>>24

00000000 00000000 00000000 11111111 255 in binary as an int

2a. Explain logical and Bitwise operators with an example (06M)

Boolean logical operators

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

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

The logical Boolean operators, &, |, and ^, operate on boolean values in the same way that they operate on the bits of an integer. The logical ! operator inverts the Boolean state: !true == false and !false == true. The following table shows the effect of each logical operation:

```
/ 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);
System.out.println(" !a = " + g);
}
}
```

Java defines several bitwise operators that can be applied to the integer types: long, int, short, char, and byte. These operators act upon the individual bits of their operands. They are summarized in the following table:

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

The Bitwise Logical Operators
The bitwise logical operators are &, |, ^, and ~. The following table shows the outcome of each operation. In the discussion that follows, keep in mind that the bitwise operators are applied to each individual bit within each operand.

The Bitwise NOT(optional)
Also called the bitwise complement, the unary NOT operator, ~, inverts all of the bits of its operand. For example, the number 42, which has the following bit pattern:
00101010
becomes
11010101
after the NOT operator is applied

The Bitwise AND
The AND operator, &, produces a 1 bit if both operands are also 1. A zero is produced in all other cases. Here is an example:
00101010 42
&00001111 15

_____
00001010 10

The Bitwise OR
The OR operator, |, combines bits such that if either of the bits in the operands is a 1, then the resultant bit is a 1, as shown here:
00101010 42
| 00001111 15

_____
00101111 47

The Bitwise XOR
The XOR operator, ^, combines bits such that if exactly one operand is 1, then the result is 1.

00101010 42
^ 00001111 15

_____
00100101 37

The Left Shift(optional)
The left shift operator, <<, shifts all of the bits in a value to the left a specified number of times. It has this general form:
value << num
Here, num specifies the number of positions to left-shift the value in value. That is, the << moves all of the bits in the specified value to the left by the number of bit positions specified by num

byte a = 64, b;
int i;
i = a << 2;
b = (byte) (a << 2);

a=64
01000000
01 00000000
a: 64
i and b: 256 0

The right shift (optional):
The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times. Its general form is shown here:
value >> num
Here, num specifies the number of positions to right-shift the value in value. That is, the >> moves all of the bits in the specified value to the right the number of bit positions specified by num.
int a = 35;
a = a >> 2; // a contains 8
Looking at the same operation in binary shows more clearly how this happens:
00100011 35
>> 2
00001000 8

The Unsigned Right Shift(optional):
The >> operator automatically fills the high-order bit with its previous contents each time a shift occurs.
int a = -1;
a = a >>> 24;
Here is the same operation in binary form to further illustrate what is happening:
11111111 11111111 11111111 11111111 –1 in binary as an int
>>>24
00000000 00000000 00000000 11111111 255 in binary as an int

```
2a. publicclass Example {
publicstaticvoid main(String[] args) {
        inta;
        for(a=0;a<3;a++) {
                intb=-1;
                System.out.println(" "+b);
                b=50;
                System.out.println(" "+b);
        }
}
```

}
For the output of the above code
Inserting another 'int b' outside the for loop, what is the output.

Output (2m)

```
-1
50
-1
50
-1
50
```

Inserting another 'int b' outside the for loop,

```
publicstaticvoid main(String[] args) {
        inta;

      int b;// case 1
        for(a=0;a<3;a++) {
                intb=-1;
                System.out.println(" "+b);
                b=50;
                System.out.println(" "+b);
        }
}
}
```

Ouput : error (1m)

```
publicstaticvoid main(String[] args) {
      inta;

      for(a=0;a<3;a++) {
            intb=-1;
            System.out.println(""+b);
            b=50;
            System.out.println(""+b);
      }
      intb;

}
}
```

Ouput : values will not change (1m)
(OR)

```
-1
50
-1
50
-1
50
```

3a. what is typecasting? Illustrate with an example, what is meant by automatic type promotion

To create a conversion between two incompatible types, we must use a cast. A cast is simply an explicit type conversion. It has this general form: (1m)
(target-type) value
For example, the following fragment casts an int to a byte. If the integer's 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.

int a;
byte b;
// ...

b = (byte) a;

Automatic Type Promotion in Expressions(2M)

➢Along with assignments, type conversions will occur even in expressions.

➢ In an expression, the precision required of an intermediate value will sometimes exceed

the range of either operand. For example,

byte a = 40;

byte b = 50;

byte c = 100;

int d = a * b / c;

The result of the intermediate term a * b exceeds the range of either of its byte operands. Tohandle this kind of problem, Java automatically promotes each byte, short, or char operandto int when evaluating an expression. This means that the subexpression a * b is performedusing integers. 2000, the result of the intermediate expression 50 * 40, is legal even though a andb are both specified as type byte.

Automatic promotions may cause compile-time errors. For example, the correct code causes a

Problem

byte b =50;

b = b * 2; // Error, Cannot assign an int to a byte

The code is attempting to store 50 * 2, a valid byte value, back into a byte variable. Because theoperands were automatically promoted to int when the expression was evaluated, the reuslt hasalso been promoted to int. Thus, the result of the expression is now of type int, which cannot beassigned to a byte without the use of a cast. This is true even if, the value being assigned wouldstill fit in the target type.During the consequences of overflow, we should use an explicit cast,

byte b = 50;

b = (byte) (b * 2);

The Type Promotion Rules (1M)

Java defines several type promotion rules that apply to expressions. They are -

1. all byte, short and char values are promoted to int.

2. if one operand is long, the whole expression is promoted to long.

3. if one operand is float, the entire expression is promoted to float.

4. if one of the operands is double, the result is double.

2M

```
class Promote {
  public static void main(String args[]) {
    byte b = 42;
    char c = 'a';
    short s = 1024;
    int i = 50000;
    float f = 5.67f;
    double d = .1234;
    double result = (f * b) + (i / c) - (d * s);
    System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
    System.out.println("result = " + result);
  }
}
```

3b. How to declare two dimensional arrays in java ? explain with an simple example.

● Syntax:

type var-name[ ] [ ];

var-name = new type [size] [size];

or

type var-name [ ] [ ] = new type [size] [size];

example

```
class TwoDArray
{
public static void main(String args[])
{
int twoD [ ] [ ] = new int [4] [5];
int i,j,k=0;
for(i=0;i<4;i++)
for(j=0;j<5;j++)
{
twoD[i][j] = k;
k++;
}
for(i=0;i<4;i++)
{
for(j=0;j<5;j++)
System.out.print (twoD[i][j] + " ");
System.out.println();
}
}
```

}

4a. Explain switchcase with an example.
The switch statement is Java's multiway branch statement. It provides an easy way todispatch execution to different parts of your code based on the value of an expression.
Here isthe general form of a switch statement:

```
switch (expression) {
case value1:
// statement sequence
break;
case value2:
// statement sequence
break;
.
.
.
case valueN :
// statement sequence
break;
default:
// default statement sequence
```

```
// A simple example of the switch.
class SampleSwitch {
public static void main(String args[]) {
for(int i=0; i<6; i++)
switch(i) {
case 0:
System.out.println("i is zero.");
break;
case 1:
System.out.println("i is one.");
break;
case 2:
System.out.println("i is two.");
break;
case 3:
System.out.println("i is three.");
break;
default:
System.out.println("i is greater than 3.");
}
}
}
```

The output produced by this program is shown here:

```
i is zero.
i is one.
i is two.
i is three.
i is greater than 3.
i is greater than 3
```

Nested switch Statements
switch as part of the statement sequence of an outer switch. This is called anested switch.

```
switch(count) {
case 1:
```

```
switch(target) { // nested switch
case 0:
System.out.println("target is zero");
break;
case 1: // no conflicts with outer switch
System.out.println("target is one");
break;
}
break;
case 2: // ...
```

summary, there are three important features of the switch statement to note:

• The switch differs from the if in that switch can only test for equality, whereas if canevaluate any type of Boolean expression. That is, the switch looks only for a matchbetween the value of the expression and one of its case constants.

• No two case constants in the same switch can have identical values. Of course, aswitch statement and an enclosing outer switch can have case constants in common.

• A switch statement is usually more efficient than a set of nested ifs

4b. Differences between for loop and for each with an example

For loop	ForEach
1. Syntax:for(initialization; condition; iteration) statement; 2. Applicable for only java Statements 3. Sequentially increment or decrement 4. It can perform both read and write	1.Syntax:for(type its-var : collection) 2. Applicable for only Array statements. 3 Only increment. 4 It can only perform write.

5a. Discuss Lexical issues in JAVA program.

Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.

- **Whitespace:** Java is a free-form language. In Java, whitespace is a space, tab, or newline.
- **Identifiers:** Used for class names, method names, and variable names. An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters. They must not begin with a number, again, Java is case- sensitive.
  - AvgTempcount, a4, $test, this_is_ok are Valid
  - 2count, high-temp, Not/ok are Invalid
- **Literals:** A constant value in Java is created by using a literal representation of it. It can be used anywhere a value of its type is allowed.
- 100, 98.6, 'X', "This is a test"

- **Comments:**Astherearethreetypesof comments defined by Java.
  1. Single comment
  2. Multiline
  3. documentation comment

  Documentation comment is used to produce an HTML file that documents your program.

  The documentation comment begins with a /** and ends with a */.

- **Separators** : The most commonly usedseparator in Java is the semicolon. As you have seen, it is used to terminate statements.

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

Java Keywords

There are 50 keywords currently defined in the Java language.

These keywords, combined with the syntax of the operators and separators, form thefoundation of the Java language.

These keywords cannot be used as names for a variable, class, or method.

The keywords const and goto are reserved but not used.

in addition to the keywords, Java reserves the following: true, false, and null.

These are values defined by Java

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

5b. with an example , explain ternary operator.

· ternary (three-way) operator that can replace certain types of if-then-else statements.
· This operator is the ?.
· The ? has this general form: expression1 ? expression2 : expression3
· Here, expression1 can be any expression that evaluates to a boolean value. If expression1 is true, then expression2 is evaluated; otherwise, expression3 is evaluated.
· The result of the ? operation is that of the expression evaluated.
· Both expression2 and expression3 are required to return the same type, which can't be void.

Largest among three numbers:

```
publicclass Ternary {

        publicstaticvoid main(String[] args) {
        int a=40,b=39,c=99,res;
         res=(a>b)?((a>c)?a:c):((b>c)?b:c);
        System.out.println(res);// 99
        }

}
```

6a. Definition of class and object in JAVA with syntax.

Class is the logical construct upon which the entire Java language is built because *it defines the shape and nature of an object*. It defines a new data type.

The General Form of a Class
A class contains data (member or instance variables) and the code (member methods) that operate on the data.
The general form can be given as:

```
class classname
{
type var1;
type var2;
.....
type method1(para_list)
{
//body of method1
}
type method2(para_list)
{
//body of method2
}
.....
}
```

Variables declared within a class are called as instancevariables because every instance (or object) of a class contains its own copy of these variables. The code iscontained within methods. Methods and instance variables collectively called as members of the class.

Object :object is an instance of a class.

Creating a class means having a user-defined data type. To have a variable of this new data type, we should create an object. Consider the following declaration:

```
Box b1;
```

This statement will not actually create any physical object, but the object name b1 can just refer to the actual object on the heap after memory allocation as follows:

```
b1 = new Box ();
```

We can even declare an object and allocate memory using a single statement:

```
Box b1=new Box();
```

6b.Write a java program to print even numbers from 0 to 100.

```
publicclass EvenNumbers {
publicstaticvoid main(String[] args) {
        System.out.println("Even numbers from 0 to 100:");

// Using a for loop to iterate from 0 to 100
for (inti = 0; i<= 100; i++) {
// Checking if the current number is even
if (i % 2 == 0) {
                System.out.println(i + " ");
            }
        }
    }
}
```