Internal Assessment Test 1 Solutions – November 2025

Sub:	Object Oriented Programming with JAVA					Sub Code:	BCS306A	Branch :	AIML/CSE AIML
Date:	**05/11/25**	Duration:	90 min	Max Marks:	50	Sem/Sec :	III A, B & C		OBE

	Answer any FIVE FULL Questions	**MARKS**	**CO**	**RBT**

1 a	**List and explain any three features of Object-Oriented Programming with example**
	Object-Oriented Programming helps organize complex programs using **objects & classes**.
	The main features are **Encapsulation**, **Inheritance**, and **Polymorphism**.
	1. Encapsulation
	• Encapsulation means **binding data and methods** into a single unit called an **object**.
	• It **protects data** from unauthorized access by using **private** access modifiers.
	• Data can be accessed only through **public methods** (getters/setters).
	class Student {
	private int marks; // data hidden
	public void setMarks(int m) { marks = m; }
	public int getMarks() { return marks; }
	}
	2. Inheritance
	• Inheritance allows one class to **acquire properties and behaviors** of another class.
	• It promotes **code reusability** and avoids duplication.
	Example:
	class Animal {
	void eat()
	{ System.out.println("Eating...");
	}
	}
	class Dog extends Animal {
	void bark()
	{ System.out.println("Barking...");
	}
	}
	3. Polymorphism
	• Polymorphism means **many forms** — the same method can behave **differently** depending on the object.
	• Achieved using **method overloading** or **method overriding**.
	Example:
	class Shape {
	void draw()
	{ System.out.println("Drawing Shape");
	}
	}
	class Circle extends Shape {
	void draw()
	{
	System.out.println("Drawing Circle");
	}
	}
		5	CO1	L2

1 b	**Discuss the primitive data types used in Java with example** • ***Primitive /Simple data types:*** Defines eight types of data: *byte, short, int, long, char, float, double, and boolean*. **Type** **Length** **Min. Value** **Max. Value** byte 8 bits -128 127 short 16 bits -32768 32767 int 32 bits -2,147,483,648 2,147,483,647 long 64 bits -9,223,372,036,854,775,808 9,223,372,036,854,775,80 float 32 bits -3.4E+38 +3.4E+38 double 64 bits -1.7E+308 +1.7E+308 boolean 1 bit Possible values are ***true*** or ***false*** char 16 bits Unicode / International character set • **Integers:** Java defines four integer types: *byte, short, int, and long*. All of these are *signed, positive and negative values*. Java does not support unsigned, positive-only integers. **Eg: *int num1, num2,...*** • **byte:** The smallest integer type, a signed 8-bit type, has a range from –128 to 127. • Useful when you're working with a stream of data from a network or file. **Eg: *byte Byte1, Byte2,...*** • **short:** A signed 16-bit type, has a range from –32,768 to 32,767. It is probably the least-used type. **Eg: s*hort s1, s2;*** • **int:** Most commonly used integer type, a signed 32-bit type that has a range from –2,147,483,648 to 2,147,483,647. • **long:** A signed 64-bit type, useful for those occasions where an **int** type is not large enough to hold the desired value. • This makes it useful when big, whole numbers are needed.	5	CO1	L1

2 a	**What do you mean by Type conversion and Type casting? Give examples**
	1. Type Conversion (Implicit Type Casting / Widening Conversion)
	• It is **automatically** done by the compiler.
	• A smaller data type is **converted into a larger data type**.
	• **No data loss** occurs.
	• Also called **Widening Conversion**.
	class TypeConversionExample {
	public static void main(String[] args) {
	int num = 10;
	double val = num; // int converted to double automatically
	System.out.println(val);
	}
	}
	Output: 10.0
	Explanation:
	Here, int (4 bytes) is automatically converted into double (8 bytes).
	
	2. Type Casting (Explicit Type Casting / Narrowing Conversion)
	• It is done **manually by the programmer**.
	• A larger data type is **converted into a smaller data type**.
	• May lead to **data loss**.
	• Also called **Narrowing Conversion**.
	Example:
	class TypeCastingExample {
	public static void main(String[] args) {
	double val = 9.78;
	int num = (int) val; // double casted to int manually
	System.out.println(num);
	}
	}
		6	CO1	L1

Output: 9

Explanation:

Here, the fractional part .78 is lost during conversion from double to int.

Extra

Type Conversion	Type Casting
Done **automatically** by compiler.	Done **manually** by programmer.
Also called **Widening conversion**.	Also called **Narrowing conversion**.
No data loss occurs.	**Data loss** may occur.
Example: `int → double`	Example: `double → int`

2 b **Explain the concept of array in Java with example** — 4 — CO1 — L2

An **array** is a collection of elements of the **same data type** stored in **contiguous memory locations**

It helps to store multiple values in a single variable and access them using **index numbers**.

1. One-Dimensional (1D) Array

➞ Used to store a **list of elements** in a single row.

dataType[] arrayName = new dataType[size];

Example:

```
class OneDArray {
   public static void main(String[] args) {
      int[] marks = {85, 90, 75, 80};
      System.out.println("1D Array Elements:");
      for(int i = 0; i < marks.length; i++)
         System.out.println(marks[i]);
   }
}
```

Output:

```
1D Array Elements:
85
90
75
80
```

2. Two-Dimensional (2D) Array

➞ Used to store **data in rows and columns** (like a table).

dataType[][] arrayName = new dataType[rows][columns];

Example:

```
class TwoDArray {
   public static void main(String[] args) {
      int[][] matrix = { {1, 2, 3}, {4, 5, 6} };
      System.out.println("2D Array Elements:");
      for(int i = 0; i < 2; i++) {
         for(int j = 0; j < 3; j++) {
            System.out.print(matrix[i][j] + " ");
         }
         System.out.println();
      }
   }
}
```

Output:

```
2D Array Elements:
1 2 3
4 5 6
```

3	**Write a program to perform stack operations using proper class and methods.**
	import java.util.Scanner;
	
	public class FixedSizeStack {
	private int maxSize;
	private int top;
	private int[] stackArray;
	
	public FixedSizeStack(**int** size) {
	maxSize = size;
	stackArray = **new int**[maxSize];
	top = -1;
	}
	
	public boolean isEmpty() {
	return (top == -1);
	}
	
	public boolean isFull() {
	return (top == maxSize - 1);
	}
	
	public void push(**int** value) {
	if (isFull()) {
	System.***out***.println("Stack is full. Cannot push " + value);
	} **else** {
	stackArray[++top] = value;
	System.***out***.println("Pushed " + value + " onto the stack.");
	}
	}
	public int pop() {
	if (isEmpty()) {
	System.***out***.println("Stack is empty. Cannot pop.");
	return -1; // Return a sentinel value to indicate an error.
	} **else** {
	int poppedValue = stackArray[top--];
	System.***out***.println("Popped " + poppedValue + " from the stack.");
	return poppedValue;
	}
	}
	public void display() {
	System.***out***.print("Stack Contents: ");
	for (**int** i = 0; i <= top; i++) {
	System.***out***.print(stackArray[i] + " ");
	}
	System.***out***.println();
	}
	
	public static void main(String[] args) {
	Scanner scanner = **new** Scanner(System.***in***);
	// System.out.print(" maximum stack size is 10: ");
	System.***out***.print("Enter the maximum stack size: ");
	int maxSize = scanner.nextInt();
	FixedSizeStack stack = **new** FixedSizeStack(maxSize);
	
	while (**true**) {
	System.***out***.println("\nStack Operations:");
	System.***out***.println("1. Push");
		10	CO2	L3

```
            System.out.println("2. Pop");
            System.out.println("3. Display");
            System.out.println("4. Quit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter the value to push: ");
                    int valueToPush = scanner.nextInt();
                    stack.push(valueToPush);
                    break;
                case 2:
                    int poppedValue = stack.pop();
                    if (poppedValue != -1) {
                        System.out.println("Popped value: " + poppedValue);
                    }
                    break;
                case 3:
                    stack.display();
                    break;
                case 4:
                    scanner.close();
                    System.exit(0);
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
```

4	**What are constructors? Give the type and explain the properties of constructor. Support with appropriate example** Constructor is a special type of member method which is invoked automatically when the object gets created. Constructors are used for object initialization. They have the same name as that of the class. Since they are called automatically, there is no return type for them. Constructors may or may not take parameters • Every class is provided with a ***default constructor*** which initializes all the data members to respective *default values*. (Default for numeric types is zero, for character and strings it is null and default value for Boolean type is false.) • In the statement *classname ob= new classname();* the term *classname()* is actually a constructor call. • If the programmer does not provide any constructor of his own, then the above statement will call default constructor. • If the programmer defines any constructor, then default constructor of Java cannot be used. • So, if the programmer defines any parameterized constructor and later would like to create an object without explicit initialization, he has to provide the default constructor by his own. For example, the above program, if we remove ordinary constructor, the statements like Box b1=new Box(); will generate error. To avoid the error, we should write a default constructor like – Box(){ } Now, all the data members will be set to their respective default values. class Box	10	CO2	L1

```
{
double w, h, d; double volume()
{
return w*h*d;
}
Box() //ordinary constructor
{
w=h=d=5;
}
Box(double wd, double ht, double dp) //parameterized constructor
{
w=wd;
h=ht;
d=dp;
}
}
```

5a **Differentiate two paradigms in programming language** — 5 — CO1 — L2

Procedure-Oriented Programming (POP)	Object-Oriented Programmi
1. Program is divided into **functions**.	1. Program is divided into **objects**.
2. Focuses on **how to do** a task (step-by-step).	2. Focuses on **what to do** using rea entities.
3. **Data and functions are separate** and may lead to data insecurity.	3. **Data and functions are combin** single unit called an **object**.
4. Difficult to maintain, modify, and reuse code.	4. Easy to **maintain, modify, and** due to modular structure.
5. Examples: C, Pascal, BASIC.	5. Examples: Java, C++, Python.

5b **Write a java program with a method to check whether given number is prime or not.** — 5 — CO1 — L3

```
import java.util.Scanner;

class PrimeCheck {
   int num;

   // Method to read number
   void getNumber() {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter a number: ");
      num = sc.nextInt();
   }

   // Method to check prime
   void checkPrime() {
      int count = 0;
      for (int i = 1; i <= num; i++) {
         if (num % i == 0)
            count++;
      }

      if (count == 2)
         System.out.println(num + " is a Prime Number.");
      else
         System.out.println(num + " is not a Prime Number.");
```

```
    }

    public static void main(String[] args) {
        PrimeCheck obj = new PrimeCheck(); // create object
        obj.getNumber();                   // read number
        obj.checkPrime();                  // check prime
    }
}
```

6a **Discuss Lexical issues in JAVA program** 6 CO2 L2

- **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.
    - AvgTemp count, 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:**As there are three types of 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 used separator 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 the foundation 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

6b	public class BOX {
	public static void main(String[] args) {
	BOX b1 = new BOX();
	BOX b2=b1;
	System.out.println(b1);
	b1=2;
	System.out.println(b1);
	}
	}
	Does above java code get compile? If provide output write the reason., if no write the reason and correct answer.
	
	Identifying Errors (1 Mark): Explain that assigning b1 = 2; is invalid because it attempts to assign an int to an object reference.
	• **Explanation of Corrected Code (1 Mark)**: Explain that b1 = new BOX(); correctly assigns a new BOX instance to b1.
	• **toString() Method Explanation (1 Mark)**: Mention that overriding the toString() method provides a more readable output when printing an object.
	• **Expected Output (1 Mark)**: State that the corrected code will output "BOX instance" twice.
		4	CO2	L3