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

Internal Assessment Test 1 – November 2024

Sub:	**Object Oriented Programming with JAVA**					Sub Code:	**BCS306A**	Branch:	**AIDS & CSE(AIDS)**
Date:	**07/11/2024**	Duration:	**90 minutes**	Max Marks:	**50**	Sem/Sec:	**III -A, B & C**		**OBE**

		Answer any FIVE FULL Questions	**MARKS**	**CO**	**RBT**
1	a	**What are the primitive data types in java programming language?** Primitive data types - includes byte , short , int , long , float , double , boolean and char.	4	1	L1

	b	**List and explain different jump statements used in java with an example for each**
		In Java, **jump statements** are used to transfer control to another part of the program. These include break, continue, and return.
		Example
		The break statement is used to exit a loop or terminate a switch statement prematurely.
		`public class BreakExample {`
		`  public static void main(String[] args) {`
		`  for (int i = 1; i <= 5; i++) {`
		`  if (i == 3) {`
		`  break; // Exits the loop when i equals 3`
		`  }`
		`  System.out.println("i: " + i);`
		`  }`
		`  }`
		`}`
		The continue statement skips the current iteration of a loop and proceeds to the next iteration.
		Example
		`public class ContinueExample {`
		`  public static void main(String[] args) {`
		`  for (int i = 1; i <= 5; i++) {`
		`  if (i == 3) {`
		`  continue; // Skips the iteration when i equals 3`
		`  }`
		`  System.out.println("i: " + i);`
		`  }`
		`  }`
		`}`
		The return statement is used to exit from a method and optionally return a value.
		`public class ReturnExample {`
		`  public static int square(int num) {`
		`  return num * num; // Returns the square of the number`
		`  }`
		
		`  public static void main(String[] args) {`
		`  int result = square(4);`
		`  System.out.println("Square of 4: " + result);`
		`  }`
		`}`
			6	1	L2

2	a	**Explain the scope and lifetime of the variable with an example**
		Scope
		The scope of a variable determines where in the program the variable can be accessed. It depends on where the variable is declared.
		Types of Variable Scopes:
		1. Local Scope: Variables declared inside a method, constructor, or block are local to that block.
		2. Instance Scope: Instance variables are declared inside a class but outside any method. They belong to the object.
		3. Class (Static) Scope: Static variables are declared with the static keyword and belong to the class, not individual objects.
		Lifetime
		The **lifetime** of a variable refers to how long the variable stays in memory:
		• **Local variables** exist until the method or block in which they are declared finishes execution.
		• **Instance variables** exist as long as the object they belong to is in memory.
		• **Static variables** exist for the lifetime of the program (or the class in memory).
		public class LocalVariableExample {
		public void display() {
		int number = 10; // Local variable
		System.out.println("Local variable: " + number);
		}
		
		public static void main(String[] args) {
		LocalVariableExample example = new LocalVariableExample();
		example.display();
		// System.out.println(number); // Error: Cannot access local variable outside its method
		}
		}
			4	1	L2

	b	**Differentiate between type conversion and type casting**
		Type Conversion
		Type conversion is an **automatic** process where the Java compiler converts one data type into another.
		Characteristics:
		• **Automatic (Implicit)**: Performed by the compiler without explicit programmer intervention.
		• **Safe Conversion**: Converts from a smaller data type to a larger data type (widening conversion).
		• **No Data Loss**: As smaller data types fit into larger data types, there's no risk of precision loss.
		Example (Type Conversion):
		public class TypeConversionExample {
		public static void main(String[] args) {
		int intNumber = 100;
		double doubleNumber = intNumber; // Automatic conversion from int to double
		System.out.println("Integer: " + intNumber);
		System.out.println("Converted to double: " + doubleNumber);
		}
		}
		Type Casting
		Type casting is an **explicit** process where the programmer forces a conversion from one data type to another.
		Characteristics:
		• **Explicit (Manual)**: Requires the programmer to specify the target data type.
		• **Risk of Data Loss**: Common when converting from a larger data type to a
			6	1	L2

		smaller data type (narrowing conversion).
		• **Potential for Errors**: May cause loss of precision or overflow issues.
		public class TypeCastingExample {
		public static void main(String[] args) {
		double doubleNumber = 100.99;
		int intNumber = (int) doubleNumber; // Manual casting from double to int
		System.out.println("Double: " + doubleNumber);
		System.out.println("Casted to integer: " + intNumber);
		}
		}			

3	a	**Compare and contrast method overloading and method overriding with a suitable example**.
		Method Overloading
		Method overloading allows a class to define multiple methods with the **same name** but with **different parameter lists**.
		Characteristics:
		• Occurs **within the same class**.
		• Distinguished by the **number or type of parameters**.
		• **Compile-time polymorphism**: The method to execute is determined during compilation.
		• Return type can differ, but it is not sufficient to distinguish methods (must differ in parameters).
		Example (Method Overloading):
		public class MethodOverloadingExample {
		// Overloaded methods
		public void display(int num) {
		System.out.println("Displaying number: " + num);
		}
		
		public void display(String text) {
		System.out.println("Displaying text: " + text);
		}
		
		public void display(int num, String text) {
		System.out.println("Displaying number and text: " + num + ", " + text);
		}
		
		public static void main(String[] args) {
		MethodOverloadingExample example = new MethodOverloadingExample();
		example.display(10); // Calls the method with int parameter
		example.display("Hello"); // Calls the method with String parameter
		example.display(20, "World"); // Calls the method with int and String parameters
		}
		}
		Method Overriding
		Method overriding allows a subclass to **provide a specific implementation** of a method that is already defined in its superclass.
		Characteristics:
			10	2	L2

		• Occurs **across different classes** in an inheritance hierarchy (superclass and subclass).
		• The overriding method must have the **same name, return type, and parameters** as the method in the superclass.
		• **Runtime polymorphism**: The method to execute is determined during runtime.
		• Requires the use of the @Override annotation (optional but recommended for clarity).
		• Access modifier of the overriding method cannot reduce the visibility of the method in the superclass.
		Example (Method Overriding):
		class Parent {
		public void show() {
		System.out.println("This is the Parent class method.");
		}
		}
		
		class Child extends Parent {
		@Override
		public void show() {
		System.out.println("This is the Child class method.");
		}
		}
		
		public class MethodOverridingExample {
		public static void main(String[] args) {
		Parent obj1 = new Parent();
		obj1.show(); // Calls the method in the Parent class
		
		Parent obj2 = new Child();
		obj2.show(); // Calls the overridden method in the Child class (runtime polymorphism)
		}
		}			

4	a	**Write a note on the following: Garbage collection b) this keyword** **Garbage Collection in Java** Garbage Collection (GC) in Java is the process of automatically identifying and reclaiming memory that is no longer used by the application. This helps manage memory efficiently and prevents memory leaks. **this keyword** The **this** keyword in Java is a reference variable that refers to the **current object** of a class. It is used primarily within an instance method or a constructor to eliminate ambiguity between instance variables and parameters or to invoke other methods or constructors in the class.	4	2	L2
	b	**Discuss the following terms with an example:** **a) Static b) final c) Abstract methods d) Abstract classes**	6	2	L2

		Static: In Java, the static keyword is used to indicate that a particular member (variable, method, block, or nested class) belongs to the class rather than any specific instance of the class. This means that the member can be accessed without creating an object of the class. **Final:** In Java, the final keyword is a modifier that can be applied to variables, methods, and classes to enforce specific behaviors. It is used to declare constants, prevent method overriding, or inheritance, ensuring immutability and stability in code. **Abstract methods**: An abstract method is a method declared in an abstract class or interface that does not have a body (implementation). It only provides the method signature (name, return type, and parameters), and the implementation must be provided by a subclass. Abstract methods are used to enforce that certain behaviors must be implemented in the subclasses, making them a key feature in achieving polymorphism**.** **Abstract classes**: An abstract class in Java is a class that is declared using the abstract keyword. It serves as a blueprint for other classes and cannot be instantiated directly. Abstract classes are used to define common properties and behaviors for a group of related classes while leaving specific details to be implemented by subclasses.			
5	a	**Define constructors in Java.** **Explain different types of constructors in Java with an example for each.** In Java, a **constructor** is a special type of method that is used to initialize objects when they are created. It is called automatically when an object of a class is instantiated. Constructors have the same name as the class and do not have a return type. The main purpose of a constructor is to initialize the object's state (i.e., its instance variables). Types of Constructors in Java: Default Constructor No-argument Constructor Parameterized Constructor Copy Constructor Constructor Overloading	10	2	L1
6	a	**Explain usage of super keyword in Java with suitable examples** In Java, the super keyword is used to refer to the immediate parent class (superclass) of the current object. It has several important uses, including: 1. Accessing Parent Class Methods 2. Accessing Parent Class Constructors	6	3	L3

		3. Accessing Parent Class Variables 4. Invoking Parent Class Constructor in Constructor Chaining			
	b	**Explain how variables in interfaces are used. Give example.** All variables declared in an interface are implicitly public, static, and final. This means: o public: The variable is accessible from outside the interface. o static: The variable belongs to the interface, not to instances of implementing classes. It can be accessed using the interface name (i.e., InterfaceName.variableName). o final: The variable is a constant and cannot be modified once assigned a value. Cannot be modified: • Since interface variables are final, they cannot be reassigned after initialization. Initialization: • Interface variables must be initialized at the time of declaration since they are final (constants).	4	3	L3

CI **CCI** **HoD**