SOLUTIONS & SCHEME IAT2
Date 24th June 2021

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

Q.No.	Solution Scheme	Marks
1a.	How do namespaces help in preventing pollution of the global namespace?	6M

Ans	Namespaces enable the C++ programmers to prevent pollution of the global namespace that leads to name clashes. The term 'global namespace' refers to the entire source code. It also includes all the directly and indirectly included header files. By default the name of each class is visible in the entire source code, that is, in the global namespace. This can lead to problems when a class with the same name is defined in two header files and both the header files are included in a program as shown below
	
	// A1.h header file
	class A
	{
	};
	
	// A2.h header file
	class A
	{
	};
	
	// cpp file
	#include "A1.h"
	#include "A2.h"
	void main()
	{
	A Aobj; // Error: Ambiguity error due to multiple definitions of A
	}
	
	To overcome the ambiguity problem, we need to enclose the two definitions of the class in separate namespaces and the corresponding namespace followed by the scope resolution operator, must be prefixed to the name of the class while referring to it in the source code as shown below
	
	// A1.h header file
	namespace A1 // beginning of namespace A1
	{
	class A
	{
	};
	} // End of namespace A1
	
	// A2.h header file
	namespace A2 // Beginning of namespace A2
	{
	class A
	{
	};
	} // End of namespace A2
	
	// cpp file
	#include "A1.h"
	#include "A2.h"
	int main()
	{
	A1 :: A A1obj; // A1obj is an object of class A defined in A1.h
	A2 :: A A2obj; // A2obj is an object of class A defined in A2.h
	}
	
	Now, the two definitions of the class are enveloped in two different namespaces. Thus, the ambiguity encountered in the first program can be overcome by using namespaces.	

1b	Guess the output and justify your answer.
	class Operator
	{
	public static void main(String[] args)
	{
	int a = 40;
	int b = 21;
	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 = 287 [2marks] Justification of each value of a,b,c,d carries [2 marks]	4M

2	Define constructor and Destructor . Explain different types of C++ constructor with example code.	10M

Ans	Constructor: is a member function of a class which is called automatically for each object created, immediately after memory has been allocated for the object.
	It appears as a member function of each class whether it is defined or not.
	[1M]
	Destructors are usually used to deallocate memory and do other cleanup for a class object and its class members when the object is destroyed.[1M]
	Types of constructor:
	
	The following are the various types of constructors -
	
	1. Default constructor
	2. Zero Argument constructor
	3. Parameterized constructor
	4. Explicit constructor
	5. Copy constructors
	
	1. Default constructor
	
	When there is no explicit constructor defined by the user, the compiler defines the constructor. That constructor is called the default constructor. The definition of default constructor will not have any statements.
	
	// Example program to illustrate default constructor
	// The user program which doesn't have a constructor
	class A
	{
	……
	……
	public:
	……
	……
	// no constructor
	};
	
	In the above program the user hasn't defined the constructor. Hence the compiler defines the default constructor as follows.
	
	class A
	{
	……
	……
	public:
	A(); // prototype inserted implicitly by compiler
	……
	……
	// no constructor
	};
	A:: A()
	{
	// empty definition inserted implicitly by compiler
	}
	
	int main()
	{
	A aobj; // The default constructor is called	

```
        …….
}
```

2. Zero-Argument Constructor:

The user can define his own constructors. If the user defines a constructor, then the compiler does not define default constructor, instead the user defined constructor will be called every time the object is created. Zero argument constructor is the constructor which doesn't take any argument. It has zero parameters.

```
// Example program to illustrate zero argument constructor

#include <iostream>
using namespace std;

class A
{
        public:
                int a,b;

                // Zero argument  constructor
                A()
                {
                        a = 10;
                        b = 20;
                }
};

int main()
{
        // Default constructor called automatically
        // when the object is created
        A aobj;
        cout << "a = " << aobj.a << endl;
        cout << "b= " << aobj.b << endl;

        return 0;
}
```

Output:

```
$ g++ zero_arg_cons.C
$ ./a.out
a = 10
b= 20
```

In the above program, the zero argument constructor initializes the data members, a and b with values 10 and 20 respectively. Hence, all objects of class A will be initialized with the values of 10 and 20 for a and b respectively.

3. PARAMETERIZED CONSTRUCTOR:

Parameterized constructors are constructors which take one or more than one argument. The arguments help to initialize an object when it is created. To create a parameterized constructor, we need to add parameters to the constructor.

```
// Example program to demonstrate parameterized constructor
#include <iostream>
using namespace std;

class A
{
        public:
                int a,b;
```

```
                    // Parameterized constructor
                    construct(int x, int y)
                    {
                            a = x;
                            b = y;
                    }
};
int main()
{
        // Parameterized constructor called
        A  aobj (10,20);
        cout << "a = " << aobj.a << endl;
        cout << "b= " << aobj.b << endl;

        return 0;
}
```

Output:

```
$ g++ ParamConstructor.C
$ ./a.out
a = 10
b= 20
```

4. Explicit Constructors:

Explicit constructors do not allow unwanted implicit type conversions. Constructors are declared as explicit constructors by prefixing their declarations with the explicit keyword. Explicit constructors can prove to be useful for the programmer if he is creating a class for which an implicit conversion by the constructor is undesirable.

```
// Example program to illustrate explicit constructors
// constructor without explicit keyword
#include<iostream>
using namespace std;

class A
{
  int data;

public:
  A(int a)
  {
        data = a;
        cout<<"value of data =" << data << endl;
  }
};


int main()
{
  A a1 = 37;

  return (0);
}
```

In the line - A a1 = 37;
The integer value 37 is implicitly converted to type A before assigning the value to object a1, by calling A(int a) constructor to create an A object from the integer value.

However, if the constructor is declared as explicit constructor as shown below.

```
#include<iostream>
using namespace std;
```

```
class A
{
 int data;

public:
 explicit A(int a)
 {
        data = a;
        cout<<"value of data =" << data << endl;
 }
};


int main()
{
 A a1 = 37;

 return (0);
}
```

Then, the implicit conversion will not take place and the compiler will throw the following error

```
$ g++ explicit.cpp
explicit.cpp: In function 'int main()':
explicit.cpp:19:10: error: conversion from 'int' to non-scalar type 'A' requested
  A a1 = 37;
        ^
```

To avoid all such confusions that happen with Implicit Conversion and Constructor Conversion, we can use explicit keyword in constructor declaration. This forces the compiler not to do an implicit conversion and throws an error for this kind of code. And thus make sures the constructor is explicitly called with the right syntax.

5. Copy Constructor:

The copy constructor is a special type of parameterized constructor which initializes an object using another object of the same class. As its name implies, it copies one object to another. It is called when an object is created and equated to an existing object at the same time.

A copy constructor has the following general function prototype:

```
ClassName (const ClassName &old_obj);
```

A Copy Constructor may be called in following cases:

1. When an object of the class is returned by value.
2. When an object of the class is passed (to a function) by value as an argument.
3. When an object is constructed based on another object of the same class.
4. When the compiler generates a temporary object.

```
// Example program to illustrate copy constructor
#include <iostream>
using namespace std;
class A
{
   public:
        int x;
        A(int a)            // parameterized constructor.
        {
                x=a;
        }
       A(A &i)              // copy constructor
        {
                x = i.x;
```

	}
	};
	
	int main()
	{
	A a1(20); // Calling the parameterized constructor.
	A a2(a1); // Calling the copy constructor.
	cout << a2.x;
	return 0;
	}
	
	Output:
	
	20
	
	[8M]	

3a	**Describe the concept of bytecode**	4M
Answer	• Bytecode is the highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM). • Bytecodes are platform-independent instructions. So Java's bytecodes are portable, that is, the same bytecode can execute on any platform containing a JVM that understands the version of Java in which the bytecodes are compiled. • Bytecodes are executed by the Java Virtual Machine (JVM). • Advantages of Java or why Java is portable and more secure? JVM was designed as an interpreter for bytecode. 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. The details of the JVM will differ from platform to platform, but all understand the same Java bytecode. • If the Java program were compiled to native code, then different versions of the same program would have to exist for each type of CPU connected to the Internet. This is not the solution, thus the execution of bytecode by the JVM is the easiest way to create truly portable programs. • Bytecode files, such as Java_CLASS files, are most often generated from source code using a compiler, like javac.	
3b	**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	
4	**Differentiate between for and foreach statements in java, Write a java program to sum only first5 elements of an array {1,2,3,4,5,6,7,8,9,10}using foreach loop,**	10

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	

5a	**Explain Type Conversion and Casting inJava**	4M

Ans	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.	

5b	**What will be the out put for fallowing statements.** **Assume a=30 , b=20, x= -23** **1. a&b** **2. a|b** **3.a^b** **4 b>>3** **5.a<<2** **6.x>>>2**	6M
Ans	1. a&b =20 2. a|b=30 3.a^b=10 4 b>>3=2 5.a<<2=120 6.x>>>2=1073741818	
6a	**differentiate between While and Do-Whileloop**	4M
Ans		

BASIS FOR COMPARISON	WHILE	DO-WHILE

General Form	while (condition) {
	statements; //body of loop
	}
		do{
		.
		statements; // body of loop.
		.
		} while(Condition);

Controlling Condition	In 'while' loop the controlling condition appears at the start of the loop.	In 'do-while' loop the controlling condition appears at the end of the loop.
Iterations	The iterations do not occur if, the condition at the first iteration, appears false.	The iteration occurs at least once even if the condition is false at the first iteration.
Alternate name	Entry-controlled loop	Exit-controlled loop
Semi-colon	Not used	Used at the end of the loop

6b	**Write a java program to find largest of three numbers**	6M

Ans	public class largest{
	public static void main(String[] args) {
	int num1 = 30, num2 = 40, num3 = 7;
	if(num1 >= num2 && num1 >= num3)
	System.out.println(num1+" is the largest Number");
	else if (num2 >= num1 && num2 >= num3)
	System.out.println(num2+" is the largest Number");	

	else
	System.out.println(num3+" is the largest Number");
	}
	}
	OutPut:
	40 is the largest Number	