Internal Assessment III – March 2019

Scheme and Solutions

Sub:	Object Oriented Concepts					Sub Code:	**17CS42**	Branch:	ISE
Date:	13-5-2019	Duration:	90 min's	Max Marks:	50	Sem / Sec:	4		OBE

Q. 1 a) State and explain the important features of Object Oriented Programming paradigm.

Encapsulation:

Encapsulation is an object-oriented programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse

Encapsulation means that the internal representation of an object is generally hidden from view outside of the object's definition. Typically, only the object's own methods can directly inspect or manipulate its fields.

Data abstraction :

Data abstraction refers to providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.

Let's take one real life example of a TV, which you can turn on and off, change the channel, adjust the volume, and add external components such as speakers, VCRs, and DVD players, BUT you do not know its internal details, that is, you do not know how it receives signals over the air or through a cable, how it translates them, and finally displays them on the screen.

Inheritance

Inheritance is a way to reuse code of existing objects, or to establish a subtype from an existing object, or both, depending upon programming language support.

In classical inheritance where objects are defined by classes, classes can inherit attributes and behavior from pre-existing classes called base classes, superclasses, parent classes or ancestor classes.

The resulting classes are known as derived classes, subclasses or child classes

Polymorphism:

Polymorphism means one name, many forms. Polymorphism manifests itself by having multiple methods all with the same name, but slightly different functionality.

There are 2 basic types of polymorphism.

Overridding, also called run-time polymorphism. For method overloading, the compiler determines which method will be executed, and this decision is made when the code gets compiled.

Overloading, which is referred to as compile-time polymorphism. Method will be used for method overriding is determined at runtime based on the dynamic type of an object.

Q.1 b) Write a C++ program to count the number of objects created

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

class Counter
{
        private:
            //static data member as count
                static int count;

        public:
            //default constructor
                Counter()
                { count++; }
                //static member function
                static void Print()
                {
                        cout<<"\nTotal objects are: "<<count;
                }
};

//count initialization with 0
int Counter :: count = 0;

int main()
```

```
{
        Counter OB1;
        OB1.Print();


Counter OB2;
        OB2.Print();

        Counter OB3;
        OB3.Print();

        return 0;
}
```

Output

```
Total objects are: 1
Total objects are: 2
Total objects are: 3
```

Q. 2 a) Explain function prototyping with example

Function prototyping is necessary in C++. A prototype describes the function's interface to the compiler. It tells the compiler the return type of the function as well as the number, type, and sequence of its formal arguments.
The general syntax of function prototype is as follows:
return_type function_name(argument_list);
For example,
int add(**int**, **int**);
This prototype indicates that the add() function returns a value of integer type and takes two parameters both of integer type.
Since a function prototype is also a statement, a semicolon must follow it.

Listing 1.22 Function prototyping

```
/*Beginning of funcProto.cpp*/
#include<iostream.h>
int add(int,int);                    //function prototype

void main()
{
  int x,y,z;
  cout<<"Enter a number: ";
  cin>>x;
  cout<<"Enter another number: ";
  cin>>y;
  z=add(x,y);                        //function call
  cout<<z<<endl;
}
int add(int a,int b)                 //function definition
{
  return (a+b);
}
/*End of funcProto.cpp*/
```

Output

```
Enter a number: 10<enter>
Enter another number: 20<enter>
30
```

Q. 2 b) How the namespace helps in preventing pollution of global namespace? Explain with example.

2.5 Namespaces

Namespaces enable the C++ programmer 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.

How can this problem be overcome? How can we ensure that an application is able to use both definitions of class A simultaneously? Enclosing the two definitions of the class in separate namespaces overcomes this problem.

```
/*Beginning of A1.h*/
namespace A1                     //beginning of a namespace A1
{
  class A
  {
  };
}                                //end of a namespace A1
/*End of A1.h*/

/*Beginning of A2.h*/
namespace A2                     //beginning of a namespace A2
{
  class A
  {
  };
}                                //end of a namespace A2
```

Listing 2.44 Enclosing classes in namespaces prevents pollution of the global namespace

```
/*Beginning of multiDef02.cpp*/
#include"A1.h"
#include"A2.h"
void main()
{
  A1::A AObj1;                   //OK: AObj1 is an object of the class
                                 //defined in A1.h
  A2::A AObj2;                   //OK: AObj2 is an object of the class
                                 //defined in A2.h
}
/*End of multiDef02.cpp*/
```

Q. 3 a) List out difference between procedure oriented program and object oriented program.

Procedure Oriented Programming	**Object Oriented Programming**	
Divided Into	In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
Importance	In POP,Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

Q. 3b) Explain function overloading with example

C++ allows two or more functions to have the same name. For this, however, they must have different signatures. *Signature of a function means the number, type, and sequence of formal arguments of the function.* In order to distinguish amongst the functions with the same name, the compiler expects their signatures to be different. Depending upon the type of parameters that are passed to the function call, the compiler decides which of the available will be invoked.

Listing 1.24 Function overloading

```
/*Beginning of funcOverload.cpp*/
#include<iostream.h>
int add(int,int);                  //first prototype
int add(int,int,int);              //second prototype

void main()
{
   int x,y;
   x=add(10,20);                   //matches first prototype
   y=add(30,40,50);                //matches second prototype
   cout<<x<<endl<<y<<endl;
}

int add(int a,int b)
{
   return(a+b);
}

int add(int a,int b,int c)
{
   return(a+b+c);
}
/*End of funcOverload.cpp*/
```

Output

```
30
120
```

Q. 4 a) What is constructor? List different types of constructors and explain default constructor & destructor with example

A constructor is a special type of member function that initialises an object automatically when it is created.

Compiler identifies a given member function is a constructor by its name and the return type.

Constructor has the same name as that of the class and it does not have any return type. Also, the constructor is always public

Types of constructors

1. Zero argument Constructor/default constructor
2. Parameterized constructor
3. Copy constructor
4. Explicit Constructor
5. Destructor is a special member function that works just opposite to constructor, unlike constructors that are used for initializing an object, destructors destroy (or delete) the object.

Example

```
#include <iostream>
using namespace std;
class HelloWorld{
public:
// default Constructor
HelloWorld(){
  cout<<"Constructor is called"<<endl;
}
//Destructor
~HelloWorld(){
  cout<<"Destructor is called"<<endl;
 }
 //Member function
 void display(){
   cout<<"Hello World!"<<endl;
 }
};
int main(){
 //Object created
 HelloWorld obj;
 //Member function called
 obj.display();
 return 0;
}
```

Q. 4 b) Explain how one can bridge two classes using friend function. Write C++ program to find the sum of two numbers using bridge friend function add( )

Friends as bridges

Friend functions can be used as bridges between two classes.

Suppose there are two unrelated classes whose private data members need a simultaneous update through a common function. This function should be declared as a friend to both the

Example 2: Addition of members of two different classes using friend Function //Bridge

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

// forward declaration
class B;
class A {
    private:
      int numA;
    public:
      A(): numA(12) { }



       // friend function declaration
       friend int add(A, B);
};

class B {
    private:
       int numB;
    public:
       B(): numB(1) { }
       // friend function declaration
       friend int add(A , B);
};

// Function add() is the friend function of classes A and B
// that accesses the member variables numA and numB
int add(A objectA, B objectB)
{
   return (objectA.numA + objectB.numB);
}

int main()
{
    A objectA;
    B objectB;
    cout<<"Sum: "<< add(objectA, objectB);
    return 0;
}
```

Output

```
Sum: 13
```

Q. 5a) Write the C++ program to get employee details (empno, ename, bsalary (initialized to 1000 by constructor) and allowance) of employee class through keyboard using the method getdata( ) and display them using method dispdata( ) on the console in the format empno, ename, bsalary, allowance

```
#include<iostream.h>

class Employee
{
       int empno;
       char ename[25];
       long bsalary;
       int allowance;

       public:
       void GetData()            //Statement 1 : Defining GetData()
       {
             cout<<"\n\tEnter Employee Id : ";
             cin>>empno;

             cout<<"\n\tEnter Employee Name : ";
             cin>>ename;

             cout<<"\n\tEnter Employee Allowance : ";
             cin>>allowance


       }

       void dispdata()            //Statement 2 : Defining PutData()
       {
             cout<<"\n\nEmployee Number : "<<empno;
             cout<<"\nEmployee Name : "<<ename;




                cout<<"\nEmployee alowance : "<<allownace;
                cout<<"\nEmployee Salary : "<<balary;
          }

   };


   void main()
   {

          Employee E;            //Statement 3 : Creating Object

          E.GetData();           //Statement 4 : Calling GetData()
          E.dispdata();           //Statement 5 : Calling PutData()

   }
```

Q. 6 a) Write the advantages of Swing over AWT

- Swing provides both additional components and added functionality to AWT-replacement components
- Swing components can change their appearance based on the current "look and feel" library that's being used. You can use the same look and feel as the platform you're on, or use a different look and feel
- Swing components follow the Model-View-Controller paradigm (MVC), and thus can provide a much more flexible UI.
- Swing provides "extras" for components, such as:
    - Icons on many components
    - Decorative borders for components
    - Tooltips for components
- Swing components are lightweight (less resource intensive than AWT)
- Swing provides built-in double buffering
- Swing provides paint debugging support for when you build your own components

Q. 6b) Explain with the syntax following
i) JLabel ii) JTextField iii) JCheckBox

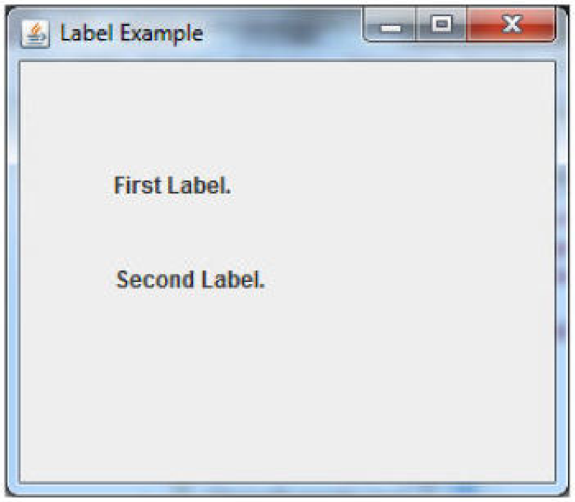
Java JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.

Syntax:

```
import javax.swing.*;
    JLabel l1,l2;
    l1=new JLabel("First Label.");
 l2=new JLabel("Second Label.");
```



Java JTextField

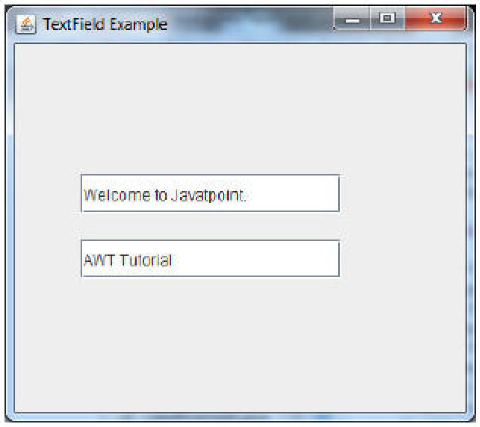
The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Syntax:

```
JTextField t1,t2;
 t1=new JTextField("Welcome to Javapoint");
 t2=new JTextField("AWT Tutorial");
```



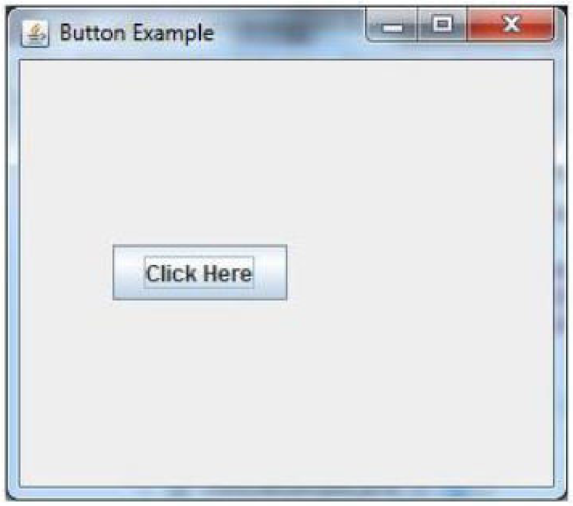
Java JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Syntax

```
JButton b=new JButton("Click Here");
```



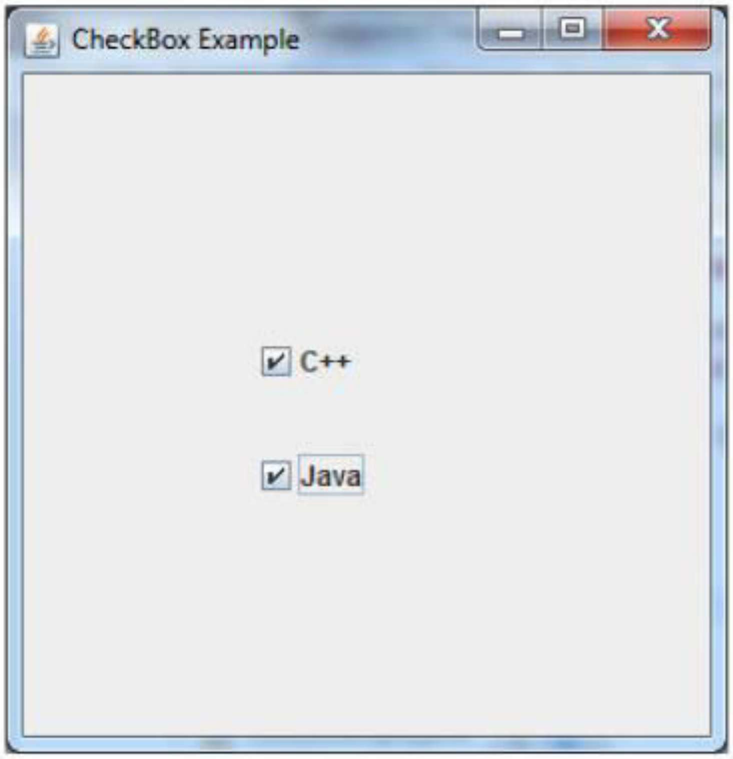
Java JCheckBox

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".

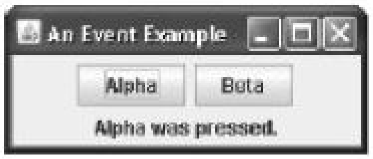
Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied

Syntax

```
JCheckBox checkBox1 = new JCheckBox("C++");
 JCheckBox checkBox2 = new JCheckBox("Java", true);
```



Q. 7 a) Create a swing Applet that has two buttons named beta and gamma. When either of the buttons pressed, it should show “beta pressed” and “gamma was pressed” respectively



```
// A simple Swing-based applet

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/*
This HTML can be used to launch the applet:

<object code="MySwingApplet" width=220 height=90>
</object>
*/

public class MySwingApplet extends JApplet {
  JButton jbtnAlpha;
  JButton jbtnBeta;

  JLabel jlab;

  // Initialize the applet.
  public void init() {
    try {
      SwingUtilities.invokeAndWait(new Runnable () {
        public void run() {
          makeGUI(); // initialize the GUI
        }
      });
    } catch(Exception exc) {
      System.out.println("Can't create because of "+ exc);
    }
  }

  // This applet does not need to override start(), stop(),
  // or destroy().

  // Set up and initialize the GUI.
  private void makeGUI() {
    // Set the applet to use flow layout.
    setLayout(new FlowLayout());

    // Make two buttons.
    jbtnAlpha = new JButton("Alpha");
    jbtnBeta = new JButton("Beta");

    // Add action listener for Alpha.
    jbtnAlpha.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent le) {
        jlab.setText("Alpha was pressed.");
      }
    });

    // Add action listener for Beta.
    jbtnBeta.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent le) {
        jlab.setText("Beta was pressed.");
      }
    });

    // Add the buttons to the content pane.
    add(jbtnAlpha);
    add(jbtnBeta);

    // Create a text-based label.
    jlab = new JLabel("Press a button.");

    // Add the label to the content pane.
    add(jlab);
   }
}
```