18CS45

Model Question Paper-1 with effect from 2019-20 (CBCS Scheme)

USN : | | | | | | | | | | |

Fourth Semester B.E. Degree Examination

Object Oriented Concepts

Time: 03 Hrs **Max. Marks:100**

Note: Answer any **FIVE** full questions, choosing at least **ONE** question from each **MODULE**

Module - 1

1 a. How do namespaces helps in preventing pollution of the global namespace. (05 marks)

b. List and explain any four features of object oriented program (05 marks)

c. Explain the following terms

i) Access specifiers

ii) Polymorphism (10 marks)

OR

2 a. Can you overload constructor and destructor? Justify with suitable example. (05 marks)

b. Define friend function. Explain how one can bridge two classes using friend Function. Write a C++ program to find the sum of two numbers using friend Function. Assume two variables are present in two different class. (10 marks)

c. What is a reference variable? Explain. Also write a program in C++ to swap two variables of type complex and display the values before and after swapping. (05 marks)

Module - 2

3 a. How " compile once and run anywhere" is implemented in Java, explain. (05 marks)

b. Write a Java program to sum only first five elements of the array using for each loop. (05 marks)

c. Explain the operations of the following operators with example.

i) % ii) >>> iii) && iv) + (concatenation operator) v) || (10 marks)

OR

4 a. List and explain Java buzzwords. (10 marks)

b. What is type casting? Illustrate with an example, what is meant by automatic type casting. (05 marks)

c. How to declare and accept values for two dimensional arrays in Java. Explain with a suitable example. (05 marks)

Module - 3

5 a. Describe the various levels of access protections available for packages and their implications with suitable examples. (10 marks)

b. Compare and contrast method overloading and method overriding with suitable example. (06 marks)

c. Explain with example, when constructors are called in class hierarchy. (04 marks)

OR

6 a. Explain usage of super keyword in Java with suitable examples. (05 marks)

b. Define Exception. Write a program which contains one method which will throw IllegalAccessException and use proper exception handlers so that exception should be printed in the calling function. (10 marks)

c. Define and explain different types of inheritance in Java. (05 marks)

7 a. What is a thread? Explain different ways of creating a thread. (05 marks)

b. With syntax explain the use of isAlive( ) and Join( ) methods. (05 marks)

c. How synchronization can be achieved between threads in Java? Explain with an example. (10 marks)

OR

8 a. Explain the role of synchronization in producer and consumer problem. (10 marks)

b. What are the differences between suspending and stopping the threads? (05 marks)

c. Explain the adaptor class with an example. (05 marks)

9 a. Discuss delegation event model with suitable example. (05 marks)

b. Explain with suitable example inner class (05 marks)

c. Explain JComboBox with example. (10 marks)

OR

10 a. Explain the following with an example for each and syntax.

i) JLabel

ii) JTextfield

iii) JButton

iv) JComboBox (10 marks)

b. Describe the two keys features of swings. (05 marks)

c. Create swing applet that has two buttons named beta and gamma. When either of the buttons pressed, it should display "beta pressed" and "gamma pressed" respectively. (05 marks)

Solutions for

Page No. 1
Date

Model Question Paper - 1

Object Oriented Concepts (18CS45)

Sem : 04 Max Marks : 100

Module - 1

1a) How do namespaces helps in preventing Pollution of the global space?

Soln: * Namespaces enable the C++ program to prevent pollution of the global namespace that leads to name clashes.

* Global namespace refers to entire source-code, by default name of the class is visible to entire source code in global namespace, this can lead to problems.

Suppose a class with same name is defined in two header files.

```
// A1.h                    // A2.h
class A                    class A
{                          {
};                         };
```

& include both these header files in a program.

```
#include A1.h
#include A2.h
void main()
{
    A obj;    // ERROR : Ambiguity error due to
}             multiple definitions of A.
```

* this problem can be overcome by Enclosing the two definitions of the class in a seperate namespaces.

```
namespace A1                    namespace A2
{                               {
    class A                         class A
    {                               {
    };                              };
}                               }
```

* Now the two definitions of a class are enveloped in two different namespaces, the corresponding namespace followed by the scope-resolution operator (::) must be prefixed to the name of the class, while refering to it

anywhere in the source code. Thus the ambiguity encountered can be overcome.

```
# include "A1.h"
# include "A2.h"
Void main()
{
    A1 :: A Obj1;  // Obj1 is an object of class A in A1.h
    A2 :: A Obj2;  // Obj2 is an object of class A in A2.h
}
```

Therefore Enclosing classes in namespaces prevents pollution of the global name space.

1b) List and Explain any four features of Object Oriented program.

Soln:
1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism.

1) Abstraction: Data abstraction refers to, providing only essential information to the outside coorld and hiding their backgroumd details. i.e to represent the needed information in program without presenting the details. Complexity can be managed.
For Ex: a database system hides certain details of how data is stored and created and maintained
* C++ classes provides different methods to the outside world without giving internal detail about these methods and data.

2) Encapsulation: Encapsulation is binding code and data together and keeps both safe from outside interference and misuse.
* corapping up of data and code inside the corapper, tightly controlled through coell defined interface.
For Ex: Classes and objects place the data and

and functions together in the same object.

3) Inheritance: Inheritance is a process of acquiring properties of another object i.e sharing common behaviour.

* one of the most useful aspect of object oriented programming is "code reusability"
* Inheritance is the process of forming a new class from an existing class. Called as "base class" & new class is formed as "derived class".
* This is a very important concept of OOP, since this feature helps to reduce the code size.

4) Polymorphism: The ability to use a method /function in different ways, in other words giving different meaning for method /functions is called "polymorphism"
* poly refers many. i.e Single method / function functioning in many ways different upon the usage is called "polymorphism"

1c) Explain (i) Access Specifier (ii) polymorphism.

1c) i) Access Specifiers:

soln: There are 3 - access specifiers.
i) private
ii) public
iii) protected.

i) private:
* Methods, variables and constructors that are declared private can only be accessed within the declared class itself.
* private access modifier is the most restrictive access level, class and interfaces cannot be private.
* Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

ii) public:
* A class, method, constructor, interface etc

declared public can be accessed from any other class.

* Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe
* Because of inheritance, all public methods and variables of a class are inherited by its subclasses

iii) protected:

* variables, methods and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members class.
* The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

ii) Polymorphism:

* The ability to use a method / function in different ways, In other words giving different meaning for method / functions is called "polymorphism".
* 'poly' refers many, morphs means forms ∴ polymorphism refers to "many forms"
* The specific action is determined by the exact nature of the situation, That is a single method / function functioning in many ways different upon the usage is called "polymorphism."

OR

2a) Can you overload the constructor and destructor? Justify with suitable example.

Soln: yes we can overload the constructor., but we cannot overload the destructor of a class in C++.

* We can have more than one constructor in a class with the same name, with different list of arguments This concept is called "constructor overloading.

* Overloaded constructors essentially have the same name and different no of arguments.
* A constructor is called depending upon the number and type of arguments passed.

Example:

```
#include <iostream>
using namespace std;
class Construct
{
public : float area;
 Construct()          // constructor with no parameter
  {
    area=0;
  }
 Construct (int a, int b)  // constructor with two
  {                          parameters
    area = a * b;
  }
 Void disp()
  {
    Cout << area
  }
};
int main()
{
 Construct O1;
 Construct O2 (10,20);
 O1. disp();
 O2. disp();
}
```

output:
0
200

Destructor : is a member function which destructs or deletes an object.

We cannot overload a destructor of a class in C++ programming.

* only one destructor / empty destructor per class should be there, it must have a void parameter list
Destructor in C++ neither takes any parameters

nor does it return anything. So multiple destructor with different signatures are not possible in a class Hence overloading is also not possible.

2b) Define friend function. Explain how one can bridge two classes using friend Function. Write a C++ program to find the Sum of two numbers using friend Function. Assume two variables are present in two different class.

Soln: * A class can have global non-member functions and member functions of other classes as "friends" Such functions can directly access the private data members of objects of the class.

* A friend function is a non-member function that has special rights to access private data members of any object of the class of whom it is a friend.

* 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 classes.

→ "Friend" keyword should appear in prototype only and not in the definition.

```
#include <iostream>
using namespace std;                    // C++ program to find
                                        Sum of two numbers using
                                        "friend" function.
Class temp
{
int a,b, Sum.
public: void read()
        {
        Cout<<" Enter the values for a & b"
        Cin >> a >> b;
        }
friend void add (temp &t);              // declaration of
                                        friend function.
```

```
class A      // Class declaration
class B      // Class definition
{
private : int b;
public : B (int n)     // constructor.
         {b=n;
         }
friend void Printvalue (A x, B y); // friend fun declaration
};

Class A
{
 private :  int a;
 public : A (int n)    // constructor.
          {
           a=n;
          }
 friend void Printvalue ( A x , B y);
};

void print value (A x, B y) // friend function defn
 {
  Cout << "The values are a= "<<x.a" and b= "<<y.b;
  Cout <<"The Sum is " << x.a + y.b;
 }.
int main()
 {
   A a1(10);
   B b1(20);
   print value (a1, b1);
 }.
```

2c) what is a reference variable ? Explain . Also write a C++ program to swap two variables of type complex and the display the values before and after swapping.

Soln: A reference variable is an alias, i-e another name for an already existing variable. When a variable is declared as a reference, it becomes

an alternative name for an existing variable. A variable can be declared as a reference by putting "&" in the declaration.

* It shares the memory location with an existing variable.

The Syntax for declaring a reference variable is as follows

<datatype> & reference var.name = existing var_name.

Ex: int &iRef = x;

'x' is an existing integer type variable & iref is reference to it.

x & iref have seperate entries in the OS., their addresses are actually same

// C++ program to swap the two variables.

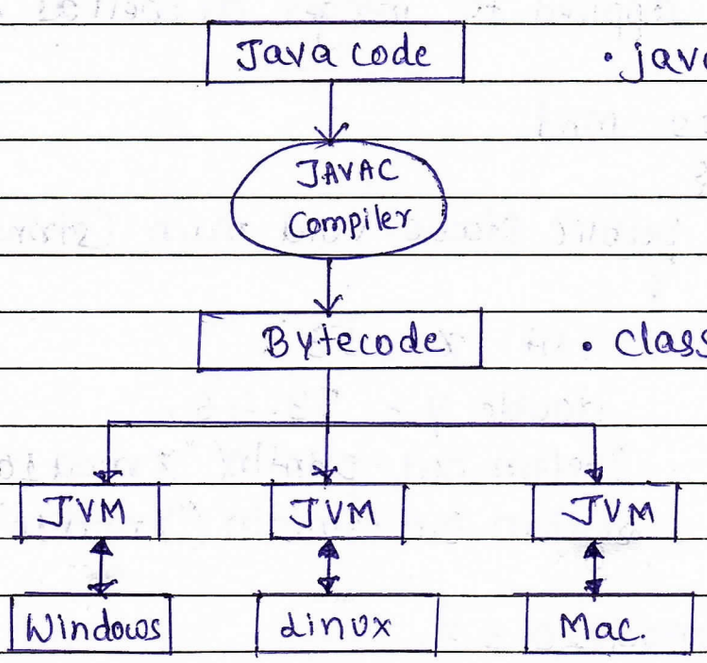
```
#include<iostream>
using namespace std;

void main()
 {
   int a, b;
   cout << "Enter two values";
   cin >> a >> b;
   cout<<"Before swapping -a="<< a <<"andb="<< b;

   swap (a, b);
   cout << "After swapping a=" <<a <<"andb=" <<b;
 }
void swap (int &x, int &y)
 {
   int temp;
   temp= x;
    x = y;
    y = temp;
 }
```

Instead of a & b, we can use 'real' and 'imag' as two variables of Complex.
just replace a, b as real & imag

Module - 2

3a) How "compile once and run anywhere" is implemented in Java. Explain.

Soln:
* Compilation of the java source code will generate the bytecode. It will create <filename>.class containing the bytecode.
* Java program can be compiled using JAVAC compiler. Java program once compiled can be run at any System.
* So Java combines both approaches : compiled and interpreted. First Java compiler translates Java code into bytecode instructions, Bytecodes are not machine instructions, therefore java interpreter generates "machine code" that can be directly executed by the machine i.e running the Java program.
* The concept "write once - run anywhere" (known as platform independent, one of the key feature of of Java Language, that makes java most powerful language. The programs written on one platform can run on any platform provided the platform must have JVM (Java Virtual Machine).
* JVM is a virtual machine that can execute Java bytecode, it is the code execution component of the Java Software platform.



3b) Write a Java Program to sum only first five elements of the array using foreach loop.

Soln:
```
// program using "foreach loop" → which enables to traverse the complete array without using index variable.
public class Test
{
public static void main (String [] args)
{
    float [] a = { 1.9, 2.9, 3.4, 3.5, 3.7 };
    float sum = 0;
    for (float ele : a)
    {
        Sum += ele;
        System.out.println ("Sum =" + Sum);
    }
}
```

3c) Explain the operations of the following operators with example.
i) % ii) >>> iii) && iv) + concatenation iv) ||

Soln: i) Modulus operator (%)
→ returns the remainder of a division operation. can be applied to integer as well as floating point types.

Ex:
```
class Mod
{
public static void main (String args[])
{
    int x = 53;
    double y = 53.46;
    System.out.println("x mod 10 =" + x % 10);
    System.out.println ("y mod 10" = + y % 10);
```

O/P:
x mod 10 = 3;
y mod 10 = 3.46;

ii) >>> (unsigned Right shift).

→ Shift operator, which always shifts zeros into the higher order bit.

Ex: int a = -1;

which sets all 32 bits to 1 in binary.

a = a >>> 24

This value then shifted right 24-bits. Filling the top 24 bits with zeros, ignoring normal sign extension.

-1 : 11111111 11111111 11111111 11111111

>>> 24

00000000 00000000 00000000 11111111

= 255 in binary as an int.

iii) && (Boolean logical AND) / Short circuit AND

- Operates on boolean values, and produces true if both operands are "True" and False in all other cases

A	B	A && B
A = False	False	False
True	False	False
False	True	False
True	True.	True

iv) '+' Concatenation operator.

'+' (plus sign) causes concatenation, i.e appending one value to another.

For Ex:

System.out.Println ("The num is =" + num);

'+' sign causes the value of num to be appended to the string that precedes it.

and then the resulting string is output. Actually 'num' is first converted from an integer into its string equivalent and then concatenated with the string that precedes it.

* We can join together as many items using '+' operator.

Ex:

```
System.out.println("a="+a+"b="+b);
```

v) || operator (logical boolean OR)
Short-circuit (OR)

- operates on boolean values, and produces False if both values of operands are False and "True" for other cases.

A	B	A || B
False	False	False.
True	False	True
False	True	True
True	True	True.

OR.

4a> List and explain Java Buzzwords.

Soln. 1) Simple
2) Secure
3) portable
4) Object oriented
5) Robust
6) Multithreaded
7) Architecture Neutral
8) Interpreted
9) High performance
10) Distributed
11) Dynamic.

1) Simple:

Java is Designed to easy to learn, If you understand the concept of Object-oriented Programming (OOP) Java could be easy to master. Java was designed to be easy for professional programmer to learn and use effectively.

2) Security:

When we download a program, there may be a risk; that code we are downloading might Contain virus, or harmful code, can cause damage to the program or gain unauthorised acces to System resources, Therefore in Java with the help of applets protection is achieved, not allowing its access to other parts of the computer, & no harm will be done & no security is breached.

3) Portability:

* The possibility of Internet may connect many different types of computers and operating Systems Connected to it. In case of applets, Same applet must be able to be downloaded and executed, by the wide variety of CPU's, OS and browsers connected to it. The Same code must work on all Computers. This mechanism is called portability.

4) Object-oriented:

In Java everything is an object. Java can be easily extended since it is based on the Object model Java is a pure object oriented language.

5) Robust:

Java is a Robust language; Java makes an effort to eliminate error sitcations, by emphasizing mainly on compile time error checking and run-time chelking Program will execute reliably in a variety of Systems robust programs was given a high priority in the design of JAVA.

6) Multithreaded:

Java supports multithreaded Programming, which allows us to write programs, that do many things simultaneously. This design feature allows developers to construct smoothly running interactive applications.

7) Architecture Neutral:

The central issue for the Java designers was the code longevity and portability. The concept of "write once run anywhere" is one of the important key feature of Java language, that makes Java as the most powerful language. The programs written on one platform can run on any platform.

8) Interpreted:

Java enables the programs by compiling into intermediate representation called "Java bytecode" This code can be executed on any System that implements JVM. Java interpreter generates m/c code that can be directly executed by the m/c i.e running the Java program.

9) High Performance:

* Java's cross-platform solutions have done so as to expense of performance, Java's bytecode was carefully designed, so that it could be easy to translate directly into native machine code. for very high performance by using just-in-time compiler.

10) Distributed:

Java is designed for the distributed environment of the internet. Java applications open and access remote objects on Internet. as easily they can do in local System.

4b) what is typecasting? Illustrate with an example. what is meant by automatic type casting?

Soln: Assigning a value of one type to a variable of another type is called as "type casting". when one type of a data is assigned to another type of variable is called as "automatic type conversion".

* Automatic type conversion will takes place if
  - two types are compatible
  - the destination type is larger than source type.

For ex: It is always possible to assign a 'int' value to a long variable. "widening conversion".

when these two conditions are met, Java will perform the conversion automatically

'int' is always large enough to hold all valid byte values, so no explicit cast statement is required.

* If there is no automatic type conversions are implicitly allowed, it is also possible to obtain a conversion between incompatible types. , defined from double to byte
* to do so you must cast, which performs explicit conversion between incompatible types.

For ex: If you want to assign an int value to a byte variable, this conversion will not be performed automatically,

. cox byte is smaller than int. Called " as "narrowing conversion", since explicitly making the value narrower, that it will fit into the target type.

TO create a conversion between two incompatible types, we must use "cast". A cast is simply an explicit type conversion. It has this Syntax

(target type) value.

```
Ex: int a;
    byte b;
    b = (byte) a; //
```

4C) How to declare and accept values for two-dimensional arrays in Java. Explain with a suitable example.

Soln: // Declaring 2D-Array:

```
int twoD [][] = new int [4][5];
```

* This allocates a 4 by 5 array and assigns it to twoD.
* twoD is matrix called as an array of arrays of int.

// To accept the values to 2D arrays in the form of rows and columns.
Allocate memory for 2D array, which specify the memory for the first (leftmost) dimension and allocate memory (rightmost) dimension seperately.

For Ex:

```
Class 2DArray
{
Public Static void main (String [] args)
{
  int twoD [][] = new int [4][5]
Scanner sc = new Scanner (System.in);
System.out.println ("Enter array elements");
// Accept the values to 2D array.
for (int i=0; i<4; i++)
{
for (j=0; j<5; j++)
{
  twoD [i][j] = sc.nextInt();
}
}
// Display 2D Array.
for (i=0; i<4; i++)
{
for (j=0; j<5; j++)
{
System.out.println (twoD [i][j]+" ");
}
System.out.println();
```

5a) Describe the various levels of access protections available for packages and their implications with suitable examples.

Soln: * Java provides many levels of protection to allow fine-grained control over the visibility of variables and methods within classes, subclasses and packages.
* Classes and packages are both means of encapsulating and containing the namespace and scope of variables and methods
* Packages act as containers for classes & other subordinate packages. Classes act as containers for data and code. The three access specifiers private, public and protected, provide a variety of ways to produce the many levels of access required by these categories. Following table summarizes

	Private	No Modifier	Protected	public
Same class	yes	yes	yes	yes
Same package Subclass	No	yes	yes	yes
same package non subclasses	NO	yes	yes	yes
Different packages Subclass.	NO	NO	Yes	yes
Different package non-Subclass.	NO	NO.	NO.	yes

* Anything declared as "public" can be accessed from anywhere.
* Anything declared "private" cannot be seen outside of the its class.
* When a member doesn't have an explicit access specification, it is visible to subclasses as well

as to other classes in the same package. This is default access.

* If you want to allow an element to be seen outside your current package, but only to classes that subclass your class directly, then declare that element protected.

For Ex :

1) package P1; // protection.java
public class protection
{
int n=1;
private int n_pri = 2;
protected int n_pro = 3;
public int n_pub=4;
public: protection()
{
S.O.P ("Base constructor+" n);
S.O.P (" n_pri = "+ n_pri + "n_pro = "+n_pro,
"n_pub=" + n_pub);
}.

2) package P1; // Derived.java
class Derived extends protection
{
S.O.P (" Derived constructor " + n);
//class only
S.O.P ("n_pro = " + n_pro);
S.O.P ("n_pub=" + n_pub);
}

3) Same Package.Java
package P1;
Class samepackage
{
SamePackage()
{
protection P = new Protection;
S.O.P ("Same package constructor");

```
        S.O.P ("n = " + P.n);
    // class only
        S.O.P ("n_pri = " +P.n_pri);
        S.O.P ("n_pro = " + P.n_pro);
        S.O.P ("n_pub = " + P.n_pub);
    }
}
```

This Example has two packages & five classes. Remember that the classes for the two different Packages need to be stored in directories named after their respective packages.
in this case P_1 and P_2.

The source for the first package defines 3-classes protection, Derived and Samepackage. The first class defines four int variables in each of the legal protection modes. The variable 'n' is declared with default protection. n_pri is private, n_pro is protected and n_pub is public.

Each subsequent class in this Example will try to access the variables in an instance of this class. The lines that will not compile due to access restrictions are commented out. Before each of these lines is a comment listing the places from which this level of protection would allow access.

The second class Derived is a subclass of protection in the same package P_1. This grants Derived access to every variable in protection except for n_pri, the private one, The third class, Samepackage is not a subclass of protection, but is in the same package and also has access to all but n_pri.

Following source code for other package P_2. The two classes defined in P_2, cover the other two conditions that are affected by accessControl The first class, protection2, is a subclass of P_1.protection. This grants access to all of P_1.protection's variables except for n_pri (because it is private) and 'n', the variable declared with default protection.

The default only allows access from within the class or the package, not extra package subclasses. Finally the class otherpackage has access to only one variable, n_pub, which was declared public.

```
Package P2;
Class protection2 extends P1.protection
  { protection2() {
     System.out.println ("derived other package constructor");
    // class or package only:
    // system.out.println ("n = " +n);
     // class only
    // System.out.println ("n_pri =" + n_pri);
       System.out.println ("n_pro = " + n_pro);
       System.out.println ("n_pub = "+ npub);
  }
}
```

This is file Otherpackage.java

```
package P2;
Class OtherPackage
  {
    otherPackage()
    {
      P1.Protection P = new P1.protection();
      S.O.P ("other package constructor");
    // class or package only
      S.O.P ("n =+" P.n);
    // class only
       S.O.P ("n_pri = " + P.n_pri);
```

```
// class, subclass or package only
    S.O.P ("n_pro = " + P.n_pro);
    S.O.P ("n_pub = " + P.n_pub);
  }
}
```

If you coish to try these two packages, here are two test files you can use.
the one for package P_1 is shown here

```
// Demo package P1;
package P1;

// Instantiate the various classes in P1;
public class Demo
{
  public static void main (String args[])
  {
    protection ob1 = new protection();
    Derived ob2 = new Derived();
    Same package ob3 = new Samepackage();

  }
}.
```

The test file for P_2 is shown next

```
// Demo Package P2;
Package P2;

// Instantiate the various classes in P2
public class Demo
{
  public static void main (String args[])
  {
    protection2 ob1 = new protection2();
    Otherpackage ob2 = new Otherpackage();

  }
}.
```

5b) Compare and contrast method overloading and method overriding with suitable example.

Soln:

	Method overloading	Method overriding
1	Method overloading is used to increase the readability of the Program.	Method overriding is used to provide specific implementation of the method i.e already provided by its superclass
2	Method overloading is Performed within class	method overriding occurs in two classes that have IS-A (inheritance) relationship.
3	In case of method overloading parameter must be different	In case of method overriding parameter must be same.
4	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run-time polymorphism.
5	In Java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

6	class overloading Example
	{
	int add (int a, int b)
	{ return (a+b);
	}
	int add (int a, int b, int c)
	{ return (a+b+c);
	}.
		class Animal.
		{
		void eat()
		{ S.O.P("eating..."); }
		class Dog extends Animal {
		void eat()
		{
		S.O.P("eating bread");
		} }

5c) Explain with example, when constructors are called in class hierarchy?

Soln: When a class hierarchy is created, in what order are the constructors for the classes, that make up the hierarchy called.

For Ex: given a Subclass B, and the Superclass A, is A's constructor called B's or vice-versa?

The answer is that, in a class hierarchy, constructor are called in order of derivation, from Superclass to Subclass. Further

Since 'super()' must be the first statement executed in a subclass constructor, this order is the same whether or not super() is used.

If super() is not used, then the default or parameterless constructor of each Superclass will be executed. The following program illustrates when constructors are executed.

```
Class A                    // Superclass
  {
    A()
    {System.out.println ("Inside A's constructor")
    }
  }
Class B extends A          // Subclass of A
  {
    B()
    { system.out.println(" Inside B's constructor");
    }
  }
Class C extends B          // Subclass of B.
  {
    C()
    { system.out.println (" Inside C's constructor");
    }
  }
```

```
class Consolidate
{
    public Static void main (String [] args)
    {
        C c = new C();
    }
}
```

Output:

Inside A's Constructor
Inside B's constructor
Inside C's constructor

We can see from the output, Constructors are called in order of derivation.

Because a Superclass has no knowledge of any Subclass, any initialization it needs to perform is seperate from and possibly prerequisite to any initialization performed by the Subclass. ∴ it must be executed first.

6a) Explain the usage of super keyword in Java with Suitable examples.

Soln: "Super" is a keyword of Java, which refers to the immediate parent class and is used inside the Subclass method definition for calling a method defined in the Superclass.

* A superclass having methods, that are private cannot be called, Only the methods which are public and protected can be called by the keyword "super". It is also used by class constructors to invoke constructors of its parent class

Syntax : super. <method-name>

Usage of Superclass:

- Super variables refer to the variable of a variable of the parent class.

- Super() invokes the constructor of immediate parent class.
- Super refers to the method of the parent class.

Ex:
```
class Employee
{
    int worktime = 8;
}
class clerk extends Employee
{
    int coorktime = 10;        // coorktime of clerk.
    void display()
    {
    System.out.println(Super.coorktime)
    }
public static void main (String args[])
{
    Clerk c = new Clerk();
    c.display();
}
}.
```

Output : 8.

Instance refers an instance variable, of the current class by default, but when you have to refer parent class instance variable, you have to use Super keyword to distinguish between parent class (here employee) instance variable and current class (here clerk) instance variable.

6b) Define Exception. Write a program which contains one method which will throw Illegal Access Exception and use proper Exception handler.

SOL): An Exception is an abnormal condition, that arises in a code sequence at run-time.

i.e An exception is a run-time error.

If a method is capable of causing an exception that it doesnot handle, it must specify this behaviour, so that callers of the method can guard themselves against that exception. We can do this by including a throws clause in the method's declaration.

* A throws clause lists the types of exceptions that a method might throw.

This is necessary for all exceptions, except those of type Error or Runtime Exception. or any of their Subclasses.

* All other exceptions that a method can throw must be declared in the throws clause.

If they are not, a compile time error will generate.

Example: First we need to declare that method throws IllegalAccessException. Second, main() must define a try/catch statement that catches this exception.

// program.

```
class ThrowsDemo
{
static void throwone() throws IllegalAccessExcepton
 {
  System.out.println ("Inside throwone");
  throw new IllegalAccessException ("demo");
 }
public static void main (String args[])
 {
   try {
        throwone();
      }catch (IllegalAccessException e)
      {
        System.out.println ("caught" +e);
      }
}.
```

6c) Define and Explain different types of inheritance in Java.

Soln: Inheritance is one of the concept of Object-oriented programming, which allows the creation of hierarchical classifications.
Inheritance represents the IS-A Relationship which is also known as parent-child relationship.

```
Class subclassname extends Superclassname
{
  methods & fields.
}
```

The "extends" keyword indicates that making new class that derives from an existing class.

Types of Inheritance in Java:

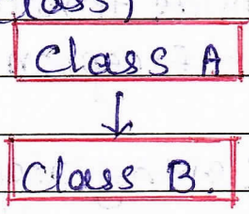
On the basis of class, there can be three types of inheritance in Java.

1) single
2) multilevel and
3) hierarchical

In Java programming, multiple and hybrid inheritance is supported through interface only.

1) Single Inheritance:

* when a class extends another class (only one class)




```
Class Dog extends Animal
{
  Void bark ()
  {
    S.O.P ("barking...")
  }
```

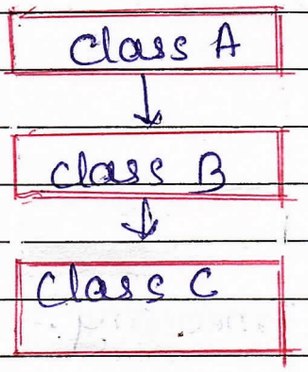
Ex:
```
class Animal
{
  Void eat()
  {
    S.O.P ("eating");
  }
}
```

```
public TestInheritance
{
  public static void main(String arg[])
  {
    Dog d = new Dog();
    d.bark();
    d.eat();
  }
}
```

output : barking
eating.

2) Multilevel Inheritance :

In multilevel Inheritance a derived class be inheriting a parent class and as well as the derived class act as parent class to other class.

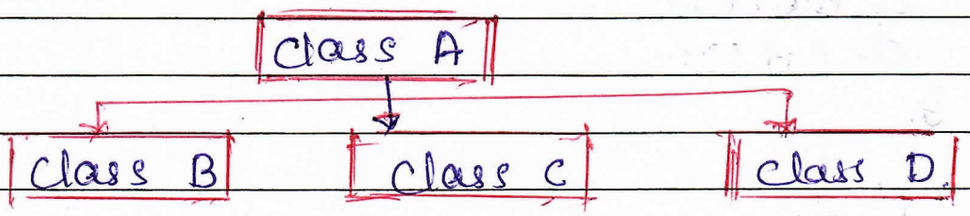



```
Class Animal
{
  void eat()
  { S.O.P("eating");
  }
}
Class Dog extends Animal
{
  void bark()
  { S.O.P("barking");
  }
}
Class Babydog extends Dog
{
  void weep()
  {
    S.O.P("weeping...");
  }
}
```

```
Class Test {
Public static void main(
    String[] args) {

BabyDog d = new BabyDo
    ()
d.weep();
d.bark();
d.eat();
```

output:
weeping
barking
eating.

3) Hierarchical Inheritance:
When two or more classes inherits a single class, it is known as "hierarchical inheritance". i.e one parent class will be inherited by many Subclasses.



Ex: Class Animal{
Void eat () {
S.O.P ("eating..."); }
}
Class Dog extends Animal{
void bark()
{ S.O.P ("barking...");
} }
Class cat extends Animal
{
Void meow()
{
System.out.println ("meowing"... };
}
Class Test
{
Public Static void main(String[]args)
{
Cat c = new cat();
C. meow();
c.cat(); C.bark // Compiletime error.
}
}
O/p: meowing
eating

Multiple & Hybrid Inheritances Not supported by Class.

Module - 4

7a) What is a thread? Explain different ways of creating a thread?

Soln. A thread is a program, can be divided into number of small processes.
Each small process can be addressed as a single thread.
In Java, Thread means, two different things.
* An instance of class (Thread class)
or thread of execution.

Creating a Thread:

We create a thread by instantiating an object of type Thread.
Java defines two ways:
1. You can implement the Runnable interface.
2. extend the Thread class.

1. Implementing Runnable:

* The easiest way to create a thread is to create a class that implements "**Runnable Interface**"

* Runnable abstracts, a unit of executable code. You can construct a thread on any object, that implements Runnable. To implement Runnable, a class need implement a single method called run which is declared like this.

```
public void run();
```

* Inside run(), define the code that constitutes the new thread, It is important to understand that run(), can call other methods, use other classes, and declare variables, just like the main thread can.

The only difference is that run(), establishes the entry point for another, concurrent thread of execution within your program. This thread will end when run() returns.

* After you create a class that implements Runnable, you will instantiate an object of type thread from within that class.
Thread defines several constructors.

Thread (Runnable threadob, String threadname);

In this constructor 'threadob' is an instance of a class that implements the "Runnable interface"

2) Extending Thread.

* Second way to create a thread is to create a new class that extends Thread and then to create an instance of that class.
* The extending class must Override the run() method which is the entry point for the new thread.
It must also call Start() to begin execution of the new Thread.
which invokes the Thread constructor:

public Thread (String threadName)

'threadname' specifies the name of the thread.

7b) With Syntax explain the use of isAlive() and join() methods.

Soln: isAlive():

* This method is defined by Thread,
final boolean isAlive()
The isAlive() method returns "true", if the thread upon which it is called is still running. It returns false "Otherwise".

while isAlive() is occasionly useful, the method that you will more commonly use to wait for a thread to finish is called join().

final void join() throws InterruptedException.

* This method waits until the thread on which it is called terminates.
* Its name comes from the concept of the calling thread waiting until the specified thread joins it.
* join(), allows us to specify, a maximum amount of time, that you want to wait for the specified thread to terminate.

7c) How synchronization can be achieved between threads in Java? Explain with an example.

Soln:* When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called "Synchronization".

* Key to synchronization is the concept of the monitor (also called "semaphore").
A monitor is an object that is used as a mutually exclusive lock or mutex. Only one thread can own a monitor at a given time.

* When a thread acquires a lock, it is said to have entered the monitor.
* All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.
These other threads are said to be waiting for the monitor. A thread that owns a monitor can reenter the same monitor if it so desires.

We can Synchronise code, in either two ways.
Both involve the use of the "Synchronised" keyword

Using Synchronized methods:

* Synchronization is easy in Java, cox all objects have their own implicit monitor associated with them.
* To enter an object's monitor, just call a method that has been modified, with the Synchronized keyword.
* while a thread is inside a Synchronized method, all Other threads, that try to call it on the same instance, have to wait.
To exit the monitor and relinquish control of the Object to the next waiting thread, the owner of the monitor simply returns from the Synchronized method.

Ex: The Synchronized Statement.

while creating Synchronized methods, within classes that you create is an easy and effective means of achieving Synchronization.
class doesn't use Synchronized methods.
Ex: the class was not created by you, but by a third party, and you do not have access to the source code.
Thus you can't add Synchronized to the appropriate methods within the class.

* Synchronized keyword involve locking and unlocking.
Synchronized method or block thread needs to acquire the lock, at this point it reads data from main memory than cache, and when it release the lock, it flushes write operation into main memory, which eliminates memory inconsistency

```
// Synchronised method

class Cse
{
  synchronized void show (String s)
  {
    System.out.println ("hello" + s);
    try {
      Thread.sleep (500);
    }
    catch (interrupted Exception e)
    {
      System.out.println ("end" +s);
    }
  }
class vdit extends threads
{
  Cse c;
  public vdit ( Cse c1, String s)
  {
    super (s);
    c = c1;
  } start();
}
public void run ()
{
  c.show (Thread.currentthread().getName());
}

};

class Test
{
  Public static void main (String args[])
  {
    Cse c1;
    vdit b = new vdit (c1, "vdit");
    vdit b1 = new vdit (c1, "csedept");
    vdit b2 = new vdit (c1, "testrun");
  }
};
```

Discuss

8a) Delegation event model with suitable example.

Soln: the modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events.

* Concept is quite simple
  a source generates an event and sends it to one or more listeners.
* The listener simply waits until it receives an event. Once an event is received, the listener processes the event and then returns.
* Adv is that the application logic that processes events is cleanly seperated from the user interface logic, that generates those events.
* A user interface element is able to "delegate" the processing of an event, to a seperate piece of code.

In the delegation event model, listeners must register, with a source in order to receive an event notification.

* This provides an important benefit:
  notifications are sent only to the listeners that want to receive them.
* This is a more efficient way to handle events than the design. ~~used~~ It eliminates overhead.

8b) What are the differences between suspending and stopping the threads?

Soln: * Suspending execution of thread: a thread which is seperate used to display the time of day. If the user doesn't want a clock, then its thread can be suspended.
* Suspend() method is used to pause the the thread.

final void suspend():
Assume,
* thread has obtained locks on critical data structures, If that thread is suspended at that point, those locks are not relenquished. Other threads that may be waiting for those resources can be deadlocked,
* once, thread suspended, it can be restarted using resume()
* Stop() method of the Thread class, is used to stop the execution of threads.
Assume that a thread is writing to a critically important datastructure and has completed only part of its changes. If that thread is stopped at that point, datastructure might be left in a corrupted state.
The thread class defines a method called "stop()" that stops a thread.
It's signature is shown here:
final void stop();
once a thread has been stopped, it cannot be restarted using resume.

8C) Explain the adapter class with an example.

Soln:
* Java provides a special feature called "adapter class", that can simplify the creation of event handlers in certain situations.
* An adapter class provides an empty implementation of all methods in an event listener interface
* Adapter Classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
* you can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which we are interested.

For Ex: the MouseMotionAdapter class has two methods,
mouseDragged() and
mouseMoved(),
which are the methods defined by the MouseMotionListener Interface.
If you were interested in only mouse drag events, then you could simply extend MouseMotionAdapter, and override mouseDragged().

The following example demonstrates an adapter. It displays a message in the status bar of an applet viewer or browser, when the mouse is clicked or dragged.

// Demonstration of adapter class.

```
import java.awt.*
import java.awt.event.*;
import java.applet.*;

<applet code="AdapterDemo" width=300 height=100>
</applet>

public class AdapterDemo extends Applet
{
  public void init()
  {
  addMouseListener( new MouseAdapter(this));
  addMouseMotionListener(new MyMouseMotion
    ~~Listen~~ Adapter(this));
  }
}
class MyMouseAdapter extends MouseAdapter
{
  AdapterDemo adapter;
  public MyMouseAdapter(AdapterDemo adapt
  {
    this.adapter = adapter;
```

```
// Handle Mouse Clicked
    public void MouseClicked (MouseEvent me)
    {
        adapter.showstatus ("Mouse Clicked");
    }
}
Class MyMouseMotionAdapter extends MouseMotionAdapter
{
    AdapterDemo a;
    public MyMouseMotionAdapter (AdapterDemo a)
        { this.a = a;
        }
    // Handle Mouse Dragged
    public void mouseDragged (MouseEvent me)
    {
        a.showstatus ("Mouse Dragged");
    }
}
```

8a) Explain the role of synchronization in producer and consumer problem.

Soln: producer-consumer problem:
It consists of four classes:
Q : the queue i.e trying to Synchronize
producer : the threaded object i.e producing queue entries
consumer : the threaded object i.e consuming queue entries.
PC : the tiny class that creates Single Q.

```
class Q
{
    int n;
    Synchronized int get()
        System.out.println ("Got" + n);
        return n;
}.
```

```
void
Synchronized put (int n) {
    { this.n = n;
    System.out.println ("put : " + n);
    }
}.
Class producer implements Runnable {
    Q q;
    producer (Q q) {
        this.q = q
        new Thread (this, "producer").start();
    }
    public void run() {
        int i = 0;
        while (true) {
            q.put (i++);
        }
    }
}
Class Consumer implements Runnable.
{
    Q q
    Consumer (Q q)
    { this.q = q
        new Thread (this."consumer").start();
    }
    public void run() {
        while (true) {
            q.get();
        }
    }
}
Class PC
{
    public static void main (string args[])
    {
        Q q = new Q()
        new Producer (q);
        new Consumer (q);
    } } System.out.println ("stop");
```

Although put() and get() methods on Q are synchronized, nothing stops the producer from overrunning the consumer, nor will anything stop the consumer from consuming the same queue value twice.

Thus we get erroneous output shown here.

```
put : 1
Got : 1
Got : 1
Got : 1
Got : 1
Got : 1
put : 2
put : 3
put : 4
put : 5
put : 6
put : 7
Got : 7
```

you can see, after producer put 1 the consumer started, & got the same 1 five times in a row. Then producer resumed & produced 2 through 7. without letting the consumer have a chance to consume them.

Module - 5

9b) Explain with suitable Example inner class.

Soln: Inner class is a class defined within another class.
or even within an expression.

Example:

Here InnerClassDemo is a top-level class that extends Applet.

* MyMouseAdapter is an inner class that extends MouseAdapter.
* Because MyMouseAdapter is defined within the scope of that class.
∴ mousePressed() method can call the showStatus() method() directly.

It no longer needs to do this via a stored reference to the applet. Thus it is no longer necessary to pass MyMouseAdapter() a reference

to the invoking object.

```
// Inner class demo.
import java.applet.*;
import java.awt.event.*;
/*
  <applet code = "InnerClassDemo" width=200 h=200
  </applet
*/
public class InnerclassDemo extends Applet
{
    public void init().
    {
      addMouseListener (new MyMouseAdapter());
    }
  Class MyMouseAdapter extends MouseAdapter
    {
    public void mousepressed (MouseEvent me)
    {
      Showstatus ("Mouse pressed");
    }
  }
};
```

9c) Explain Jcombobox with Example.

Soln: * Swing provides a combobox (a combination of a text field and drop-down list) through the JComboBox class.

* A combobox normally displays one entry, but it will also display a dropdown list that allows a user to select a different entry.

* We can create a combobox that lets the user enters a selection into the text field. The JComboBox constructor used by the example is shown here.

JComboBox (Object [] items)

'items' is an array that initializes the combobox.

passing an array of items to be displayed in the drop-down list. Items can be dynamically added to the list of choices via the addItem() method

void addItem(Object obj)

Here 'obj' is the Object to be added to the Combobox
This method must be used only with mutable Combobox.

1. JComboBox generates an action event when the user selects an item from the list.
2. JComboBox generates an action event when the state of selection changes. which occurs. when an item is selected or deselected.

Thus changing a selection means that two item events will occur.
One for the deselected item.
and another for the selected item.

One way to Obtain the item selected in the list is to call getSelectedItem() on the comboBox shown here:

Object getSelectedItem(); you will need to cast the returned value into the type of Object stored in the list.

Example:

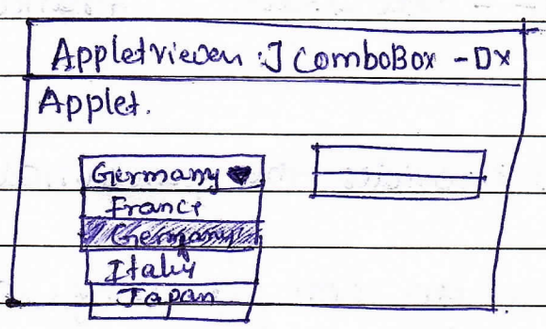
The following Example generates the combobox. ComboBox contains entries for "France", "Germany" "Italy" and "Japan".
when a country is selected, an icon-based label is updated to display the flag for that country

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

```
/*
<applet code = "JComboBoxDemo" width=300 height=200>
</applet>
*/
public class JComboBoxDemo extends JApplet {
  JLabel jlab;
  ImageIcon france, germany, italy, japan;
  JComboBox jcb;

  String flags[] = { "France", "Germany", "Italy", "Japan" };

  public void init()
  {
    try {
      SwingUtilities.invokeAndWait(
        new Runnable() {
          public void run()
          {
            makeGUI();
          }
        });
    }
    catch (Exception ex)
    {
      System.out.println("Can't create " + ex);
    }
  }
private void makeGUI()
  {
    // Change to flowLayout
    setLayout(new FlowLayout());

    // Instantiate a comboBox and it to the contentpane
    jcb = new JComboBox(flags);
    add(jcb);
    // Handle Selections
  jcb.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
```

```
String s = (String) jcb.getSelectedItem();
jlab.setIcon(new ImageIcon(s+".gif"));
};
});
```

```
// create a label and add it to the content pane
jlab = new JLabel(new ImageIcon("france.gif"));
add(jlab);
}
}
```



OR

10a) Explain the following with an example for each and Syntax.
i) JLabel
ii) JTextfield
iii) JButton.

Soln: **JLabel:**

* JLabel is Swing's easiest-to-use component. It creates a label and
* JLabel can be used to display text and/or an icon.
* It is a passive component in that it does not respond to user input.
* JLabel defines several constructors.
  JLabel(Icon icon)
  JLabel(String str).
  JLabel(String str, Icon icon, int align);

Here 'str' and icon are the text and icon used for the label.
The 'align' argument specifies the horizontal alignment of the text and / or icon within the dimensions of the label.
LEFT, RIGHT, CENTER, LEADING, or TRAILING.
These constants are defined in the "Scoing Constant" Interface.

Ex: create a label:

```
JLabel j1 = new JLabel ("France");
```

2> JButton:

* JButton class provides the functionality of a push button.
* JButton allows an icon, a string, or both to be associated with the push button.
* JButton (Icon icon)
  JButton (String str)
  JButton (String str, Icon icon)
* Here 'str' and 'icon' are the string & icon passed, & icon used for the button.
* When the button is pressed, an "ActionEvent" is generated.
  Using the ActionEvent Object passed to the actionPerformed() method of the registered ActionListener, you can obtain the action command string associated with the button.
* By default, this is string is displayed inside the button.
* We can set the action command by calling setActionCommand() on the button.
* We can obtain the action command by calling getActionCommand on the event object.

String getActionCommand() on the eventobject

String getActionCommand()

The action command identifies the button. Thus when using two or more buttons within the same application the action command gives, an easy way to determine which button was pressed.

Ex: JButton jb = new JButton (France)
jb.setActionCommand ("France");
jb.addActionListener (this);
add(jb);

3) JTextField

* 'JTextField' is the simplest scoing text component. It is also probably is most widely used text Component.
* JTextField allows you to edit one-line of text. It is derived from JTextComponent, which provides the basic functionality common to scoing text Components.
* JTextField uses the document interface for its model JTextField constructors are given beloco.

JTextField (int cols)
JTextField (string str, int cols);
JTextField (String str)

Here 'str' is the string to be initially presented, and 'cols' is the number of columns in the text field. If no string is specified, the text field is initially empty.
If the no of Columns is not specified, the text field is sized to fit the Specified string.
* JTextField generates events en response to user interaction

For Ex:
an ActionEvent is fired when the user presses ENTER.
. A caretEvent is fired each time the caret (the cursor position changes.
* To obtain the text currently in the textField call getText().

```
Ex.  jtf = new JTextField(15)
     add(jtf);
     jtf.addActionListener(new ActionListener()
     {
       public void actionPerformed(ActionEvent ae)
       {
         showStatus(jtf.getText());
       }
     }
```

10b) Describe two key features of Swings:

Soln: Two key swing features are
1. Lightweight component
2. Pluggable look and feel.

1. Swing components are Light weight:
* This means that they are written entirely in Java, and do not map directly to platform-specific peers.
* Because Lightweight components are rendered using graphics primitives, they can be transparent, which enables non rectangular shapes.
* Thus light weight components are more efficient and more flexible, & do not translate into native peers.
* Each component will work in a consistent manner across all platforms.

2. Swing Supports a pluggable Look and Feel:
* Because each swing component is rendered by Java code rather than by native peers. The look and feel of each component is determined by swing, but not by the underlying Operating System.
i.e look and feel of a component is under the control of swing.
* This fact means that it is possible to seperate the look and feel of a component from the logic of the component.
* It is possible to "plug-in" a new look and feel for any given component without creating any side effects in the code that uses that component. Look-and-feel is simply "plugged in", once this is done, all components are automatically rendered using that style.

10) Create a swing applet that has two buttons named beta and gamma. when either of the buttons pressed it should display 'beta' is pressed and "gamma" is pressed respectively.

Soln:
```
// Swing Based Applet:
import javax.swing.*
import java.awt.*;
import java.awt.*;

/* This HTML can be used to launch the applet:
<object code = "MySwingApplet" width 220 height=90>
</Object> */

public Class MySwingApplet extends Applet
{
    JButton jbtnAlpha;
    JButton jbtnBeta;
    JLabel jlab;
```

```
// Initialise the applet.
public void init() {
  try {
    SwingUtilities.invokeAndWait(new Runnable()
    { public void run()
        { makeGUI(); } });
  catch (Exception e)
    {
      System.out.println("Cant create bcoz of "+e);
    }
}
// This applet doesn't need to Override start(), stop() or
destroy
// Set up & initialize the GUI.
private void makeGUI()
{
  setLayout(new FlowLayout()); // Set the applet
  JBtn Alpha = new JButton("Alpha"); to use flowLayout
  JBtn Beta = new JButton("Beta"); // two buttons
  // Add Action listener for Alpha.
  jbtnAlpha.addActionListener(new ActionListener()
  { public void ActionPerformed(ActionEvent ie)
    {
      jlab.setText("Alpha was pressed");
    }
  });
  // Add action listener for Beta.
  jbtnBeta.addActionListener(new ActionListener()
  { public void actionPerformed(ActionEvent ie)
      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);
}
}
```

—x—