SHARIF

Part-II

Chapter-1

1. What is object oriented programming? What are the benefits of object oriented programming?

Object oriented programming (oop) is a programming example based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. For example, a person is an object which has certain properties such as height, genden age etc.

Benefits of oop

* ~~We can build the programs~~

1. Re-usability: It means reusing some facilities rather than building them again and again. This is done with the use of a class. We can use in in number of times as per our need.

2. Data Redundancy:

This is a condition created at the place of data storage (Database) where the same piece of data is held in two separate places. So the

data redandancy is one of the greatest advantages of oop. If a su user wants a similar functionality in multiple classes, he/she can go ahead by writting common class definitions for similar functionalities and inherit them.

3. Code maintenance: রক্ষণাবেক্ষণ

This feature is more of a necessity for any programming languages. It helps users from doing rework in many ways: It is always easy and time saving to maintain and modify the existing codes by incorporating new changes into them. একত্রিতকরা

4. Security: With the use of data hiding and abstraction mechanism, we are filtering out limited data to exposure, which means we are maintaining security and providing necessary data to view. প্রকাশকরা

5. Design benefits:

The object oriented programs forces the designers to have a long and extensive design phase, which results in better designs and fewer flaws. ব্যাপক

6. **Better productivity :**

OOP leads to more work done, finishing a better program, having more inbuilt features, easier reading writing and maintaining.

7. **Easy troubleshooting :**

Some problems any developers face in their work, which later becomes so brainstorming for the developers to look where the error is. By using OOP language, we will know where to look for. This is the advantage of using encapsulation.

8. **Polymorphism Flexibility :** Polymorphism is flexible and helps developers in a number of ways.

- It's simplicity
- Extensibility.

9. **Problems solving :**

OOP is specialized in this behaviour, as it breaks down your software code into bite sized - one object at a time. The broken components can be reused in solutions to different other problems (be less and more complex.)

2. What is procedure-oriented programming? Write down the characteristics of procedure-oriented programming?

Procedure oriented programming is the conventional way of programming where an application problem is viewed as a sequence of steps. In procedure oriented programming the problem is broken down into various modules.

The procedure oriented programming is the traditional approach of programming for developing application software.

High level languages like FORTRAN, COBOL, Pascal, BASIC and C etc are based on the procedure oriented approach, and consequently are called procedural languages.

Characteristics:

- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.

- Data move openly around the system from function to function.
- Functions transform data ~~from~~ from one form to another.
- Employs top-down approach in program design.
  বিশ্লেষণাত্মক

3. Write down the difference between procedure oriente programming and object oriented programming?

Procedural Oreiented programming	object orientend
1) The program is divided into small parts called functions.	(1) The program is divided into small parts called objects
2) Follows a top-down approach.	(2) follows a bottom-up approac
3) It does not have any proper way of hiding data so it is less secure.	(3) It provides data hiding so it is more secure.
4) Overloading is not possible	(4) Overloading is possible

POP	OOP
(5) There is no concept of data hiding and inheritance.	(5) The concept of data hiding and inheritance is used.
(6) The function is more important than data.	(6) Data is more important than function.
(7) It is based on the unreal world.	(7) It is based on the real world.
(8) It is used for designing medium sized program.	(8) It is used for designing Large and complex programs.
(9) It uses the concept of procedure abstraction.	(9) It uses the concept of data abstraction.
(10) Example: C, FORTRAN, Pascal, Basic, etc.	(10) Examples: C++, Java, Python, C#, etc.

3. Write down the advantages and disadvantages of object oriented programming and procedure oriented programming.

Disadvantages of OOP:

- The length of the programmes developed using OOP language is much longer than the procedural approach. Sine the program becomes longer in size, it requires more time to be executed that leads to slowen execution of the program.
- We cannot apply OOP everywhere as it is not a universal language. It is applied only when it is required. It is not suitable for all types of problem.
- Programmers need to have brilliant designing skill and programming skill along with proper planning because using OOP is little bit trickly.
- OOPs take time to get used to it.
- Everything is treated as object in OOP so before applying it we need to have excellent thinking in terms of object.

Advantages of procedure oriented programming:

- Procedural Programming is excellent for general purpose programming.
- The coded simplicity along with ease of implementation of compilers and interpreters.
- The source code is portable
- The code can be reused in different parts of the program, without the need to copy it.
- Through Procedural Programming technique, the memory requirement also slashes
- The program flow can be tracked easily.

Disadvantages:

- The program code is harder to write when Procedural Programming is employed.
- The procedural code is often not reusable.
- Difficult to relate with real world object.

5. Write down the features of object oriented programming.

- Emphasis is on data, rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows bottom-up approach in program design.

Applications of oop

- Real-time systems.
- Simulation and modeling.
- Object oriented databases.
- Hypertext, hypermedia and expertext.
- AI and expert systems.

- Neural networks and parallel programming.
- Decision support and office automation systems.
- CIM/CAM/CPD systems.

- Briefly explain the data abstraction, encapsulation, polymorphism and dynamic binding.

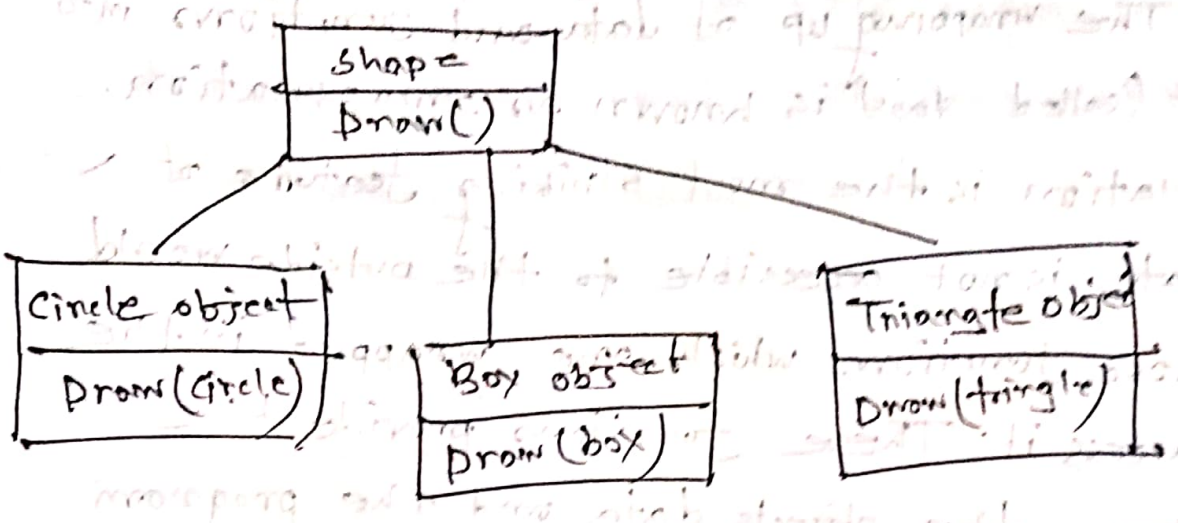
Data Abstraction: Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight, and cost, and functions to operate on these attributes. They encapsulate all the essential properties of the objects that are to be created. The attributes are sometimes called data members because they hold information.

Since the classes use the concept of data abstraction, they are known as Abstract Data Types (ADT).

Encapsulation : The wrapping up of data and functions into a single unit (called class) is known as encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. These functions provide the interface between the objects data and the program. This insulation of the data from direct access by the program is called data hiding or information hiding.

Polymorphism: Polymorphism, a Greek term, means the ability to take more than one form. An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.

For example: consider the operation of addition. For two numbers, the operation will generate sum. If the operands are strings, then the operation would produce a third string by concatenation. The process of making an operator to exhibit different behaviours in different instances is known as operator overloading.



ploymorphism

Dynamic Binding: Binding refers to the linking of procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run-time.

It is associated with ploymorphism and inheritance

□ Define class, object, method

Class: Object:

Objects are the basic run time entities in an object oriented system. They may represent a person, a place, a bank account, a table of

data or any item that the program has to handle. They may also represent user defined data such as vectors, time and lists. Program objects should be chosen such that they match closely with the real-world object.

When a program is executed, the objects interact by sending message to one another. For example, if "customer" and "account" are two objects in a program, then the customer object may send a message to the account object, requesting for the bank balance.

Class: We know that objects contain data, and code to manipulate that data. The entire set of data and code of an object can be made a user defined data type with the help of a class. In fact, objects are variable of the type class. Once a class has been defined we can create any number of objects belonging to the class.

For example: mango, apple and orange are members of the class fruit.

Method: A method is a code block that contains a series of statements. A program causes the statements to be executed by calling the method and specifying any required method arguments.

In C#, every executed instruction is performed in the context of a method.

SHAIRE

Chapter 2 and 3

2. Briefly discuss about the structure of C++ language?

A C++ program consists of the following parts:

- Documentation
- Preprocesson statements
- Global Declarations
- The main() function
  - Local Declarations
  - Program statements and Expressions.
- User defined Function

☐ Documentation section: The documentation section comprises a set of comment lines giving the name of the program, the authen and other details.

☐ Preprocesson statements: #include<iostream> is a preprocesson directive. It tells the processor to include the contents of the iostream header file in the program before compilation.

☐ Global Declaration: The variables which are declared outside of all the function and accessible ~~from~~ to all functions including main function are known as Global variables.

- The main() function: The main() is the main function where program execution begins. Every C++ program must contain only one main function.

- User defined function: Function in c++ is a unique named block of code to perform certain tasks. All the statements written in the function body are executed when a function is called by its name.

Example:

```
/*
 * File: main.cpp
 * Author: Sharif
 * Created on 01 November, 2022
 */
#include <iostream>

int main()
{

}
```

Q. Briefly describe the basic data type in c++ programming Language.

All variables use data-type during declaration to restrict the type of data to be stored. C++ supports a wide variety of data types and the programmer can select the data type appropriate to the needs of the application.

C++ supports the following data types:

1. Primary or Built in data type.
2. Derived data type.
3. User defined data type.

1. Primitive data types: These data types are built-in or predefined data types and can be used directly by the user to declare variables. Example:

- Integer: The keyword used for integer data types is int. It requires 4 bytes of memory space.
- Character: Character data type is used for storing character. The keyword used for the character data type is char. and requires 1 byte for memory space.
- Booleam: Booleam data types is used for storing booleam or logical values.

- Floating point: It is used for storing single precision floating-point values or decimal values. The key word used for the floating point data type is float and it requires only 4 bytes of memory space.

- Double floating point: Double floating point data type is used for storing double precision floating point values or decimal values. The key word used for the double floating point data type is double. and it requires only 8 bytes of memory space.

- void: Void means without any value. Void data type represents a valueless entity.

Derived data Types: The data types that are derived from the primitive or built in data types are reffered to as Derived Data types. Example:

- Function: A function is generally defined to save the user from writing the same lines of code again and again for the same unit. All the lines of code are put together inside a single function.

Syntax

FunctionType FunctionName (parameters)

☐ Annay: An annay is collection of items stored at continuous memory locations.

Syntax: DataType ArrayName [size_of_array];

☐ Pointers: Pointers are symbolic representation of addresses. They enables programs to simulate call by refference as well as to crete and manipulate dynamic data structures.

Syntax: datatype *var_name;

☐ Reference: When a variable is declared as reference, it becomes an alternative name for an existing variable. A variable can be declared as reference by putting "&" in the declaration.

User-defined Data Types:

The data types that are defined by the user are called the derived user defined data type:

☐ Class: The building block of C++ that leads to Object-Oriented Programming is a class. It is a user defined data type, which holds its own data members and member functions.

Syntax:

keyword ← class ClassName ← user defined name

```
{
    Access specifier;  // can be private, public or protected
    Data members;      // variables to be used
    Member functions()
};
```

□ Structure: A structure is a user defined data type in c/c++. A structure creates a data type that can be used to store group items of possibly different types into a single type.

Syntax;

```
struct address {
    char name[50];
    char street[100];
    char city[50];
    char state[20];
    int pin;
};
```

☐ Union: Like structures, union is user defined data type. In union all members share the same memory location

Example: both x any y share the same location. If we change x, we can see the changes being reflected in y.

☐ Enumeration: Enumeration (or enum) is a user defined data type in c. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

Syntax:

```
enum State { working=1, Failed=0};
```

☐ What is token? Briefly explain the token in C++.

A token is the smallest element of a program that is meaningful to the compiler.

1. Keywords
2. Identifiers
3. Constants.
4. Strings.
5. Special Symbols
6. Operators.

☐ Keywords: keywords are pre-defined or reserved words in a programming Language. Each keyword is meant to perform a specific function in a program. Since keywords are reffered names for a compiler, they can't be used as variable names.

Example: auto, break, continue, int, double, char, float, void, while, if etc.

☐ Identifiers: Identifiers are used as the general terminology for the naming of variables, functions, and arrays.

- They must begin with a letter or underscore (_).
- They must consist of only letters, digits, or undersore. No other special character is allowed.
- It should not be a keyword.
- It must not contain white space.
- It should be up to 31 characters long as only the first 31 characters are significant.

☐ Constants: Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined. Constants refer to fixed values. They are also called literals.

Constants may be integer, real, floating, Octal, Hexadecimal character and strings type.

☐ **Strings:** Strings are nothing but an array of characters ended with a null character ('\0') This null character indicates the end of the string. Strings are always enclosed in double-quotes.

```
char string[20] = {'S', 'h', 'a', 'r', 'i', 'f', '\0'};
```

☐ **Special symbols:**

Brackets ([]), Parentheses (), Braces {}, comma (,), colon (:), semicolon (;), Asterisk(*) etc are special symbol

☐ **Operator:**

☐ Unary operators: Those operators that require only a single operand to act upon are known as unary operators. Example : Increment and decrement.

☐ Binary operator : Needs two operand.

(1) Arithmetic
(2) Relational
(3) Logical
4. Assignment
5. Bitwise

□ What do you mean by operator overloading in C++?

Operator overloading is a manner in which OOP systems allow the same operator name or symbol to be used for multiple operations.

A simple example of this is the "+" sign.

Chapter – 5 SHARIF

1. Define object and class.

Class: A class in c++ is the building block that leads to object oriented programming. It is a user defined data type which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A c++ class is like a blue-print for an object.

The general form of a class declaration is

```
class class-name
{
  private:
     variable declantion;
     function declaration;
  Public:
     variable declaration;
     function declaration;
};
```

Object:

3. What is object ? How they created, Explain with example.

এখানে প্রথমে object এর ধারণা :

Create an object in c++ :

Before we create an actual object, it is necessary to have its class to be already created. A class is like a blueprint, and an object will be created using this class. Once a class has been created, we can create variables of that type by using the class name. For example, an class name is 'item'.

Then, item x; // memory for x is created.

creates a variable x of type item. In c++, the class variables are known. are known as objects. Therefore, x is called an object of type item.

```
class item
{
---
---.
} x;
```

Example:

```
#include <iostream>
using namespace std;
class Cellphone
{
private:
    string brandName = "Samsung";
    string model = "Galaxy";

public:
void details()
{
cout<<" cellphone details are: " <<endl;
cout<<" Brand Name :" <<brandName <<endl;
cout <<" Model Name :" <<model <<endl;
}
};

int main()
{
Cellphone obj; // created a object of type Cellphone
return 0;
}
```

2. What is the difference between Class and Structure?

Features	Structure	Class
Definition	A structure is a grouping of variable of various data types refference by the same name.	In C++, a class is defined as a collection of related variables and functions contained within a single structure.
Basic	If no access specifier is specified, all members are set to 'public.'	If no access specifir is specified, all members are set to 'private',

Declaration	struct struct_name {
	type struct_number 1;
	type struct_number 2;
	type struct_number n;
	};
		class class name {
		data member;
		member function;
		};

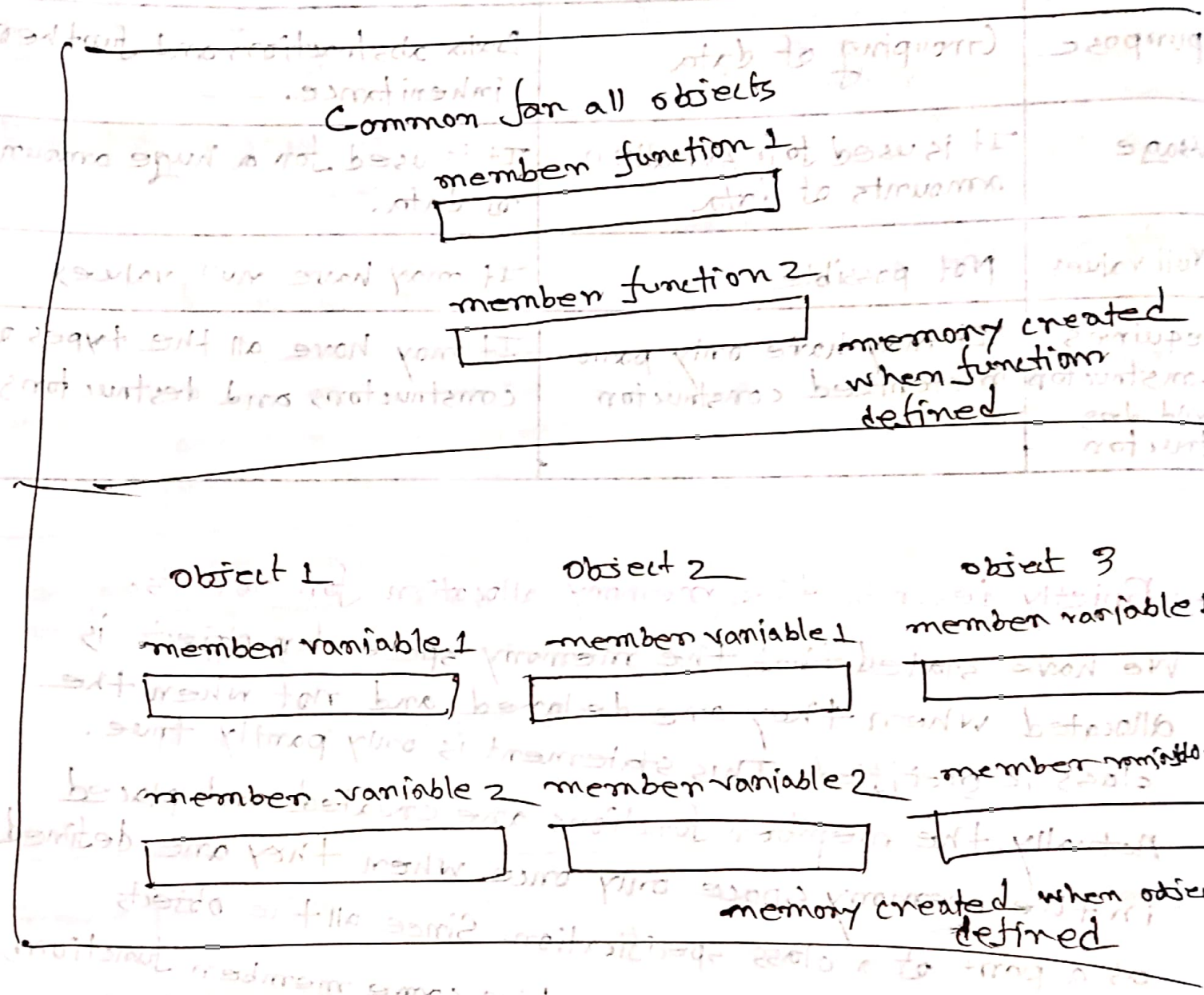
Instance	Structure instance is called the 'structure variable'	A class instance is called object.
Inheritance	It doesnot support it	It supports inheritance
memory allocated	Memory is allocated on the stak	Memory is allocated on the heap.

Features	Structure	Class
Nature	Value type	Reference type
purpose	Grouping of data	Data abstraction and further inheritance.
Usage	It is used for smaller amounts of data	It is used for a huge amount of data.
Null values	Not possible	It may have null values.
Requires constructor and destructor	It may have only parameterized constructor	It may have all the types of constructors and destructors.

6. Briefly describe the memory allocation for objects.

We have stated that the memory space for objects is allocated when they are declared and not when the class is specified. This statement is only partly true. Actually, the member functions are created and placed in the memory space only once when they are defined as a part of a class specification. Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created. Only space for ~~memory~~ member variables is allocated separately for each object. Seperate memory locations for the objects are essential

because the member variables will hold different data values for different objects.



8. What do you mean by array of objects. Explain with a suitable example:

We know that an array can be of any data type including struct. Similarly, we can also have arrays of variables that are of the type class. Such variables are called arrays of object.

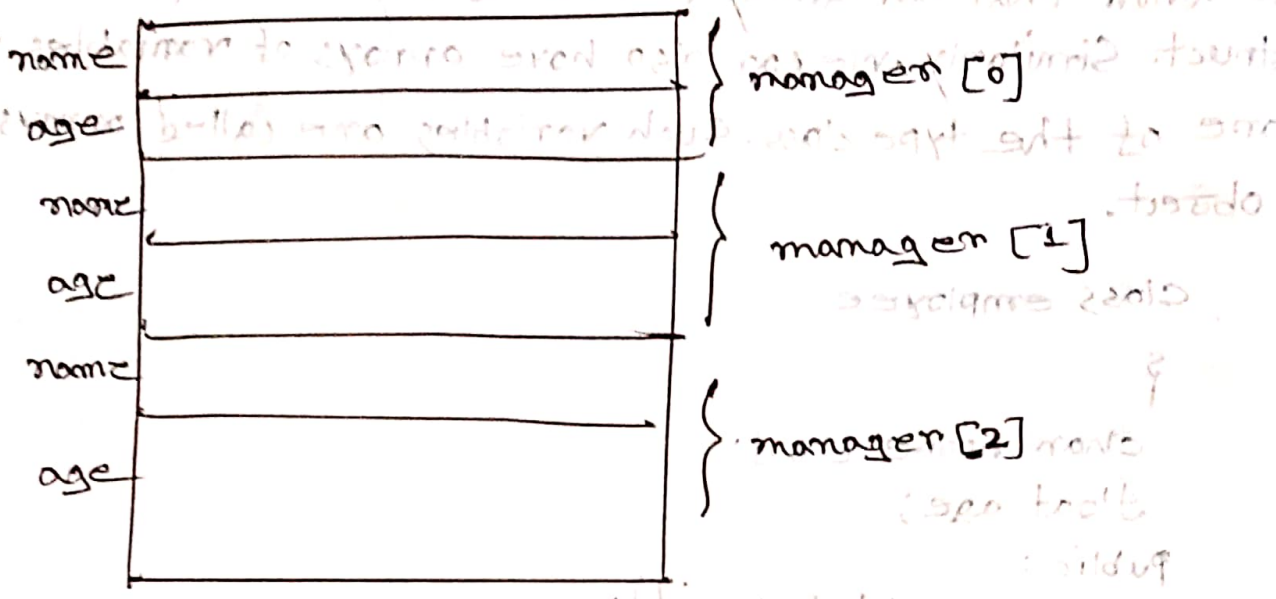
```
class employee
{
 char name[30];
 float age;
public:
    void getdata(void);
    void putdata(void);
};
```

The identifier employee is a user defined data type and can be used to create object that relate to different categories of the employee. Example

```
employee manager[3];    // array of manager
employee foreman[15];   // array of foreman
employee worker[75];    // array of worker;
```

The array managers contains three objects (managers) namely, manager[0], manager[1] and manager[2] of type employee class. Similarly, the foreman array contains 15 objects (foreman) and worker array contains 75 objects (workers).

An array of object is stored inside the memory in the same way as a multi-dimentional array.




```
# include <iostream>
  using namespace std;
  class employee
   {
    char name [30];
    float age;
    Public:
       void getdata (void);
       void putdata (void);
   };
  void employee :: getdata (void)
   {
     cout << "Enter name : ";
```

```
cin >> name;
cout << "Enter age : ";
cin >> age;
}
void employee :: putdata(void)
{
cout << "Name : " << name << endl;
cout << " Age : " << age << endl;
}
const int size=3;  // Global variable
int main()
{
employee manager[size];
for(int i=0; i<size; i++)
{
cout << "\n Details of manager " << i+1 << endl;
manager[i].getdata();
}
cout << "\n";
for(i=0; i<size; i++)
{
cout << "\n manager " << i+1 << "\n";
manager[i].putdata();
}
```

```
return 0;
}
```

What is friendly function. Write down the characteristics of friendly function.

The function that are declared with the keyword friend are known as friend function. A function can be declared as a friend in any number of classes. A friend function, although not a member function, has full access rights to the private members of the class.

```
class ABC
{
  ------
  ------
Public:
  ------
  friend void xyz(void); // declaration
};
```

Special characteristics:

- It is not in the scope of the class to which it has been declared as friend.

- Since it is not in the scope of the class, it cannot be called using the object of that class.
- It can be invoked like a normal function without the help of any object.
- Unlike member functions, it cannot access the member names directly and has to use an object name and dot membership operator with each member name.
- It can be declared either in the public or the private part of a class without effecting its meaning.
- Usually it has the objects as a arguments.

10. What do you mean by pointer to member of a class?

It is possible to take the address of a member of a class and assign it to a pointer. The address of a member can be obtained by applying the operator & to a "fully qualified" class member name. A class member pointer can be declared using the operator ::* with the class name.

For example, given the class

```
Class A
{
private:
    int m;
Public:
    void show();
};
```

We can define a pointer to the member m as follows

```
int A::*ip = &A::m;
```

The ip pointer created thus acts like a class member in that it must be invoked with a class object. In the statement above, the phrase A::* means "pointer to member of ~~class~~ A class".

The phrase ~~&A::m~~ &A::m means the "address of the m member of A class."

SHARIF

Chapter – 6

1. Define constructors and destructors with suitable example.

Constructor: A constructor is a 'special' member function whose task is to initialize the objects of its class. It is special because its name is the same as the class name. The constructor is invoked whenever an object of its associated class is created. It is called constructor because it constructs the values of data members of the class.

A constructor is declared and defined below:

```
class integer
 {
  int m, n;
    public:
       integer (void);    // constructor declared
       - - -
       - - -
 };

 integer :: integer (void)  // costructor defined
  {
   m=0; n=0;
  }
```

The declaration not only creates the object but also initializes its data members.

Destructors: A destructor, as the name implies, is used to destroy the objects that have been created by a constructor. Like a constructor, the destructor is a member function whose name is the same as the class name but is preceded by a ~~tilde~~ tilde

It is defined as shown below

```
~ class-name ( ) { }
```

Example:

```
#include<iostream>
using namespace std;
class Test
{
Public:
    Test ( )
    {
        cout<<"\n Constructor executed";
    }
   ~ Test ( )
    {
        cout<<"\n Destructor executed";
    }
};
main( )
{
Test t;
return 0;
}
```

Output

Constructor executed
Destructor executed

2. What are the special characteristics of the constructor functions?

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and therefore, and they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.
- Like other C++ functions, they can have default argumer
- Constructors cannot be virtual
- We cannot refer to their addresses.
- An object with a constructor cannot be used as a member of a union.
- They make 'implicit calls' to the operators new and delete when memory allocation is required.

3. Explain parameterized constructors with suitable example.

The constructor, initializes the data members of all the objects to zero. However, in practice it may be necessary to initialize the various data elements of different objects with different values. when they are created. C++ permits us to achieve this objective by passing arguments to the constructor function. The constructors that can take arguments are called parameterized constructors.

```
class integer
{
  int m, n;
    public:
      integer(int x, int y); // parameterized constructor
      - - -
};

integer:: integer (int x, int y)
{
  m=x; n=y;
}
```

Example:

```
#include<iostream>
using namespace std;
class point
{
  int x,y;
  public:
    point (int a, int b)    //—
    {
      x=a;
      y=b;
    }
    void display()
    {
      cout<<"("<<x<<","<<y<<")"<<"\n";
    }
};
int main()
{
  point P1 (1,1);    //—
  point P2 (5,+0);
  cout<<" Point P1= ";
  P1.display();
  cout<<" point P2= ";
  P2.display();
  return 0;
}
```

The output of program

Point P1 = (1,1)
Point P2 = (5,20)

4. Explain default constructor and overload constructor.

Default constructor: Default constructor is the constructor which doesn't take any argument. It has no parameters It is also called a zero argument constructor.

```
# include <iostream>
using namespace std;
class construct {
   public:
      int a, b;
      construct()  // Default constructor
          {
           a=10;
           b=20;
          }
    };
int main()
  {
   construct c;     // Default constructor called automatically
                    // when the object is created
   cout << " a: " << c.a << endl << "b: " << c.b;
   return 0;
}
```

Output

a: 10
b: 20

Overloaded Constructor: Constructor overloading in C++ allows a class to have more than one constructors that have the same name as that of the class but differs only in terms of a number of parameters or parameters datatype or both.

Example:

```
#include<iostream>
 using namespace std;
class construct
   {
     public:
       float area;
       construct()  // constructor with no parameter
       {
        area=0;
       }
       construct(int a, int b)  // with two parameter
       {
         area= a*b;
       }
       void display()
         {
          cout<<area<<endl;
         }
};
```

```
int main()
{
    // construct overloading
    // with two different constructors
    // of class name

    construct o;
    construct o2(10,20);

    o.display();
    o2.display();

    return 0;
}
```

Output

```
0
200
```

SHARIF

chapter-6

6. What do you mean by dynamic initialization of object? why do we need to do this?

Dynamic initialization of object refers to ~~the~~ initializing of objects ~~using dynamic constructors~~ the objects at a run time i.e. the initial value of an object is provided during run time. It can be achived by using constructors and by passing parameters to the constructors. This comes in really handy when there are multiple constructors of the same class with different inputs.

```
#include<iostream>
using namespace std;
class geeks {
    int* ptr;

  Public:
    geeks()  // Default constructor
    {
    ptr = new int; // Dynamically initializing ptr using new
    *ptr = 10;
    }
    void display()      // Function to display the value of ptr
    {
    cout << *ptr << endl;
    }
};
```

```
int main ( )
  {
    Geeks obj1;
    obj1.display();
    return 0;
  }
```

Output:

10

Dynamic initialization of objects is a technique used in many programming languages to manage initialization and initialization dependencies.

☐ Briefly describe the copy constructor with suitable example.

A copy constructor is a member function that initializes an object using another object of the same class.

In simple terms, a constructor which creates an object by initializing it with an object of the same class, which has been created previously is known as a copy constructor.

```
classname (cons classname & old_obj);
```

Example:

```
#include <iostream>
using namespace std;
class code
 { int id;
```

```
public:
    code() {}    // constructor

    code(int a)
      {
       id= a;
      }

    code(code & x)
      {
       id= x.id;
      }

    void display (void)
      {
       cout << id;
      }
};

int main()
  {
  code A(100); // object A is created and initialized
  code B(A);   // copy constructor called
  code c=A;    // copy constructor called again
  code D;      // D is created, not initialized
  D=A;         // copy constructor not called

  cout<<"\n id of A :"; A.display();
  cout<<"\n id of B :"; B.dispay();
```

```
    cout<<"\n id of c :"; c.display();
    cout<<"\n id of D:"; D.display();
     return 0;
}
```

Output:

```
id of A:100
id of B:100
id of C:100
id of D:100
```

SHARIF

Chapter – 7

1. What do you mean by operator overloading in c++ programming language ?

```
# include<iostream>
  using namespace std;
  class space
   {
    int x,y,z;
    Public:
      void getdata(int a, int b ;int c)
        {
         x=a;
         y=b;
         z=c;
        }
        void display (void)
         {
         cout<<x<<" ";
         cout<<y<<" ";
         cout<<z<<endl;
        }
        void operator– ()
         {
          x=-x;
          y=-y;
          z=-z;
        }
   };
```

```
int main()
{
  space s;
  s.getdata(10,-20,30);
  cout<<"S : ";
  s.display();
  return 0;
  -s;
  cout<<"S : ";
  s.display();
  return 0;
}
```

Output:

S: 10 -20 30

S: -10 20 -30

SHARIF

Java Graphics Programming

1. Drawing Methods of the graphics class:

Method	Description
clearRect()	Erase a rectangular area of the canvas.
copyArea()	Copies a rectangular area of the canvas to another area.
drawArc()	Draws a hallow arc
drawLine()	Draws a straight line
drawOval()	Draws a hollow oval.
drawPolygon()	Draws a hollow polygon (बहुभुज)
drawRect()	Draws a Hollow rectangle.
drawRoundRect()	Draws a Hollow rectangle with rounded corners.
drawString()	Displays a text string.
fillArc()	Draws a filled arc.
fillOval()	Draws a filled oval.
fillPolygon()	Draws a filled polygon.
fillRect()	Draws a filled rectangle
fillRoundRect()	Draws a filled rectangle with rounded corners.
getColor()	Retrieves the current drawing color
getFont()	Retrieves the currently used font.

getFontMetrics() ⟶ Retrieves information about the current font.

setColor() ⟶ Sets the drawing colour

setFont() ⟶ Sets the front.

2. Write a Java program for drawing lines and rectangulam.

```
import java.awt.*;
import java.applet.*;
public class LineRect extends Applet
{
  public void paint (Graphics g)
  {
    g.drawLine(10,10,50,50);
    g.drawRect (10,60,40,30);
    g.fillRect (60,10,30,80);
    g.drawRoundRect (10,100,80,50,10,10);
    g.fillRoundRect (20,110,60,30,5,5);
    g.drawLine (100,10,230,140);
    g.drawLine (100,140,230,10);
  }
}
<APPLET
    CODE = LineRect.class
    WIDTH = 290
    HEIGHT = 200>
</APPLET>
```

4. Write down three arguments to draw polygon for using the drawPolygon() method of graphics class.

Polygons are shapes with many sides. A polygon may be considered a set of lines connected together. The end of the first line is the beginning of the second line, the end of the second line is the beggining of the third and so on. We can draw a polygon with n sides using the drawLine() methods in n times in succession.

We can draw polygons more conveniently using the ~~Line~~ drawPolygon() method of Graphics class. This methods take three arguments.

- An array of integers containing x condinates.
- An array of integers containing y condinates.
- An integer for the total number of points.

It is obvious that x any y arrays should be of the same size and we must repeat the first point at the end of the array for closing the polygon.

```
public void paint (Graphics g)
{
int xPoints [ ] = { 10,170,80,10};
int yPoints [ ] = { 20, 40,140, 20};
int nPoints [ ] = xPoints.length;
g.drawPolygon (xPoints, yPoints, nPoints);

}
```

☐ Write a Java program for drawing polygons.

```
import java.awt.*;
import java.applet.*;
public class Poly extends Applet
{
    int x1[]={20,120,220,20};
    int y1[] = {20,120,20,20};
    int n1 = 4;
    int x2[] ={120,220,220,120};
    int y2[] ={120,20,220,120};
    int n2=4;
    public void paint (Graphics g)
    {
      g.~~drawline~~ drawPolygon(x1,y1,n1);
      g.fillPolygon(x2,y2,n1);
    }
}
```

6. Briefly describe the AWT package.

The abstract Window Toolkit (AWT) package in Java enables the programmers to create GUI based applications. It contains a number of classes that help to implement common Windows-based task, such as manipulating windows, adding scrol bars, buttons, list items, text boxes etc. All the classes are contained in the java.awt package. These classes are hierachically arranged inside the awt package. ~~in such a manner~~

AWT provides support for both standard and applet windows.

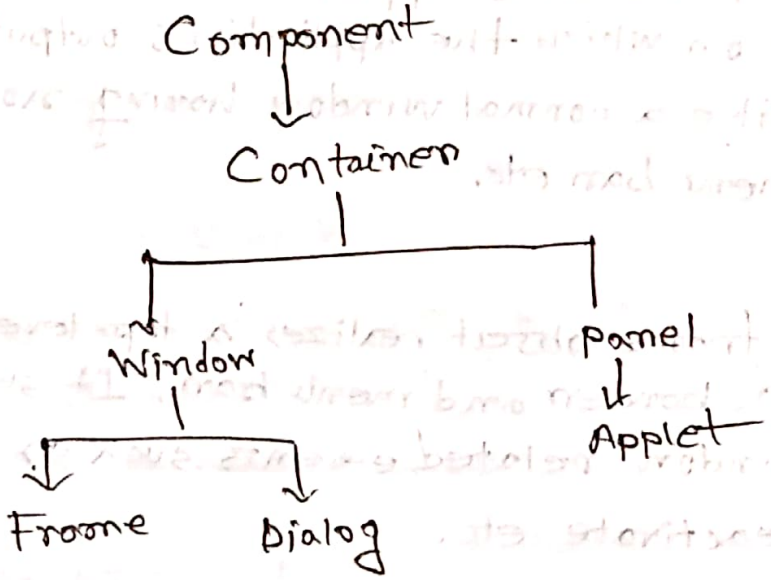


Fig: AWT Hierarchy

classes are hierarchically arranged in the awt package.

Component: Component class is the super class to all the other classes from which various GUI elements are realized. It is primarily responsible for effecting the display of a graphic object ~~of~~ on the screen. It also handles the various keyboard and mouse events of the GUI application

Container: The container object contains the other awt components. It manages the layout and placement of the various awt components within the container. A container object can contain other containers objects as well; thus allowing nesting of containers.

Window: The window object realizes a top-level window but without any border or menu bar. It just specifies the layout of the window.

Pannel: The super class of applet, Panel represents a window space on which the applications output is displayed. It is just like a normal window having no border, title bar, menu bar etc.

Frame: The frame object realizes a top-level window complete with border and menu bar. It supports common window related events such as close, open, activate, deactivate etc.

About swings (JApplet, JFrame, JButton, JTree, JComboBox etc.)

Swing is a GUI toolkit that facilitates the creation of highly interactive GUI applications. Swing is more flexible and robust when it comes to implementing graphical components. One of the main differences between swing and awt is that swing will always generate similar type of output irrespective of the underlying platform. AWT, on the other hand, is more dependent on the underlying operating system for generating the graphic components.

- Some of the key swing classes are:
  - **JApplet:** An extension of Applet class, it is swings version of applet.
  - **JFrame:** An extension of java.awt.Frame class, it is the swings version of frame.
  - **JButton:** Helps to realize a push button in swing.
  - JTabbedPane: Helps to realize a tabbed pane in swing.
  - JTree: Helps to realize a hierarchical (tree) structure in swing.
  - JComboBox: Helps to realize a combo box in swing.

Difference between AWT and Swing:

AWT	Swing
Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications
The components of Java AWT are heavy weighted	The components of Java Swing are light weighted.
Java AWT has comparatively less functionality as compared to swing.	Java Swing has more functionality as compared to AWT.
The execution time of AWT is more than swing	The execution time of swing is less than AWT.
The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
MVC pattern is not supported by AWT	MVC pattern is supported by Swing
AWT provides comparatively less powerful components	Swing provides more powerful components.

How is Java's co-ordinate system organized ?

The Java 2D API maintains two coordinate spaces.

- **User space** : The space in which graphics primitives are specified.

- **Device space** : The co-ordinate system of an output device such as a screen, window or a ~~pointer~~ printer.

User space is device independent logical co-ordinate system, the co-ordinate space that your program uses. All geometries passed into Java 2D rendering routines are specified in user space co-ordinates.

When the default transformation from user space to device space is used, the origin of user space is the upper left corner of the components drawing area. The x co-ordinate increases to the right, and the y co-ordinates increases downward. The top-left corner of a window is 0,0. All co-ordinate are specified using integers. which is usually sufficient. However, some cases require floating point or even double precision which are also supported.

Device space is a device dependent co-ordinate system that varies according to the target rendering device.

Although the co-ordinate system for a window or screen might be very different from the co-ordinate system of a printer, these differences are invisible to Java programs.

SHARIF

C++

Chapter - 8

1. What do you mean by inheritance in c++ ?

The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of object oriented programming.

Inheritance is a feature or process in which, new classes are created from the existing classes. The new class created is called "derived class" or "child class" and the existing class is known as the "base class" or "parent class". The derived class now is said to be inherited from the base class.

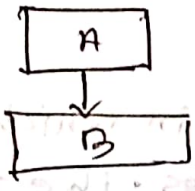
2. What do you mean by abstract class ?

An abstract class is one that is not used to create object. An abstract class is designed only to act as a base class (to be inherited by other classes). It is a design concept in program development and provides a base upon which other classes may be built.

```
class Test    // An abstract class
{
    public:
        virtual void show()=0;  // pure ventual function
};
```

☐ Briefly describe the single inheritance in C++ programming language?

Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.



Where 'A' is the base class, and 'B' is the derived class.

Syntax:

```
class subclass-name : acces_mode base-class
{
  // body of subclass
};
```

OR

```
class A
{
-----
};

class B : public A
{
-----
};
```

Example:

```
#include <iostream>
using namespace std;
class vehicle {
    public:
        vehicle()
            {
            cout << "This is vehicle" << endl;
            }
    };
class car: public vehicle {          // single inheritance
    };
int main()
    {
     car obj;
     return 0;
    }
```

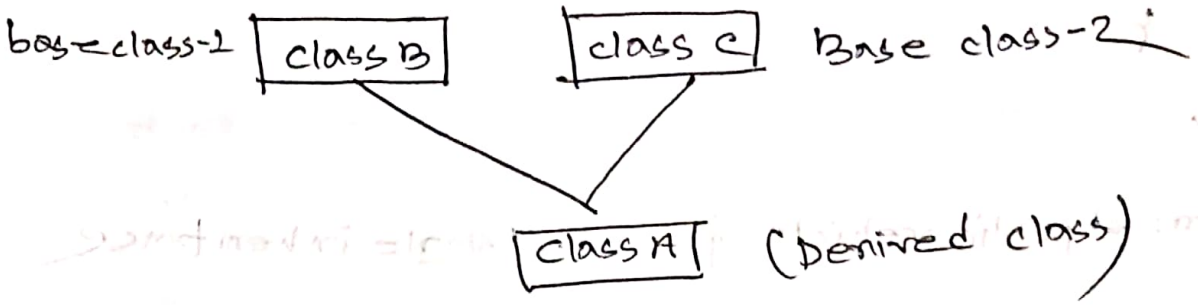
Output

This is vehicle.

4. Briefly describle the different types of inheritance in C++ programming language.

1. Single inheritance.

2. Multiple Inheritance : multiple inheritance is a feature of C++ where a class can inherit from more than one class. i.e. one subclass is inherited from more than one base class.



Syntax:

```
class sub-class-name : access_mode base_class1, access_
                       mode base_class2, ----
{
 // body of subclass
};
```

Example:

```
#include<iostream>
using namespace std;
class vehicle () {        // first base class
    public:
        vehicle()
          {
          cout<<" This is vehicle<<endl;
          }
};
```

```
class FourWheeler {          // second base class
    public:
        FourWheeler ()
        {
        cout<<"This is a 4 wheeler vehicle"<<endl;
        }
    };

class car: public vehicle, public FourWheeler {    //sub class
    };

int main()
{
    car obj;
    return 0;
}
```

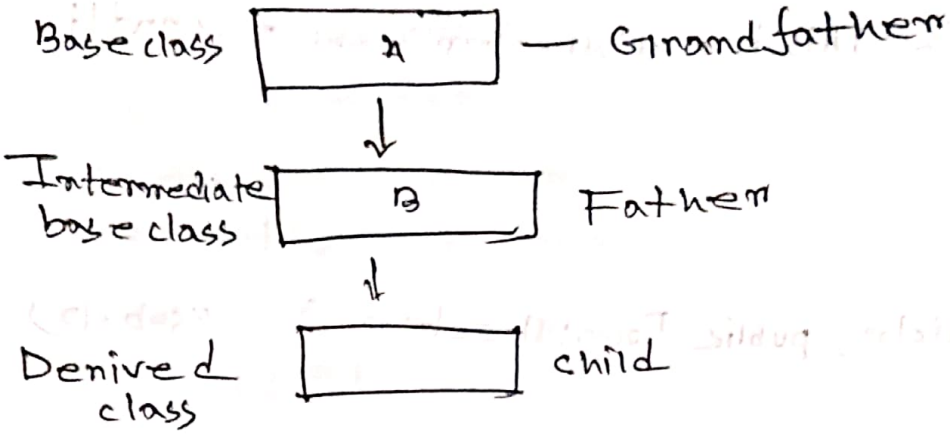
Output:

This is a vehicle

This is a 4 wheeler vehicle.

3. Multilevel Inheritance :

In this type of inheritance, a derived class is created from another derived class.



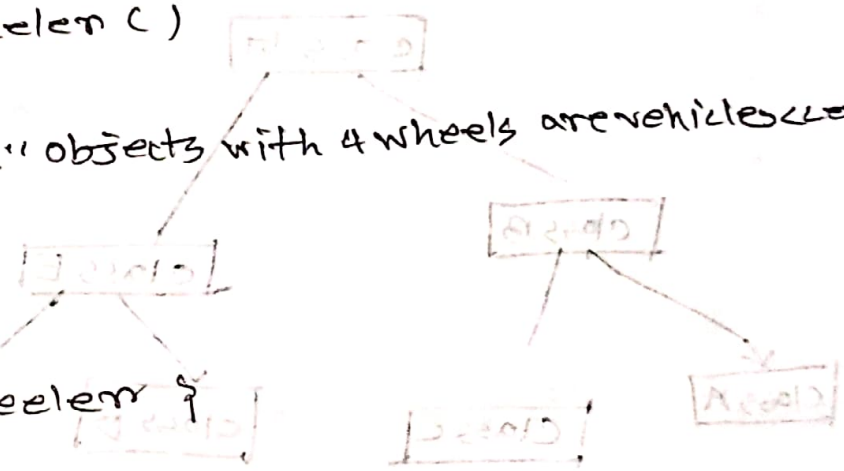
The class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. The class B is known as intermediate base class since it provides a link for the inheritance between A and C.

syntax :

```
class A
  {
  ----
  };
class B : public A
  {
  ----
  };
class C : public B
  {
  ----
  };
```

Example :

```
#include<iostream>
using namespace std;
class vehicle {
        public:
            vehicle () { cout<< "This is vehicle"<<endl; }
        };
class fourwheeler : public vehicle {
        public:
            fourwheeler ()
            {
            cout<<" objects with 4 wheels are vehicles"<<end
            }
    };
class car : public fourwheeler {
        public:
            car() { cout<<" Car has 4 wheels"<<endl;
            }
    };
int main ( )
    {
    car obj;
    return 0;
}
```



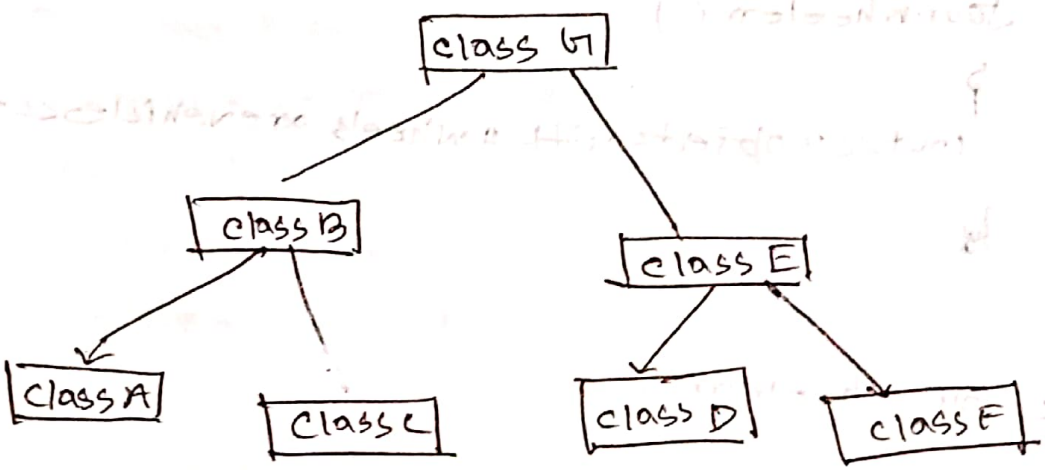
Output

This is a vehicle

Objects with 4 wheels are vehicles

Can has 4 wheels

4. Hierarchical Inheritance: In this type of inheritance more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.



Syntax:

```
class A
  {
   // body of the class A
  };
class B : public A
  {
   // body of class B.
  };
```

```
class C : public A
    {
      // body of the class C
    };
class D : public A
    {
      // body of the class D
    };
```

Example:

```
#include <iostream>
 using namespace std;
 class vehicle {          // base class
       public:
          vehicle()
            {cout << "This is vehicle." << endl;
            }
       };
class car: public vehicle {     // First sub class
            };
class bus: public vehicle {     // second sub class
          };
```

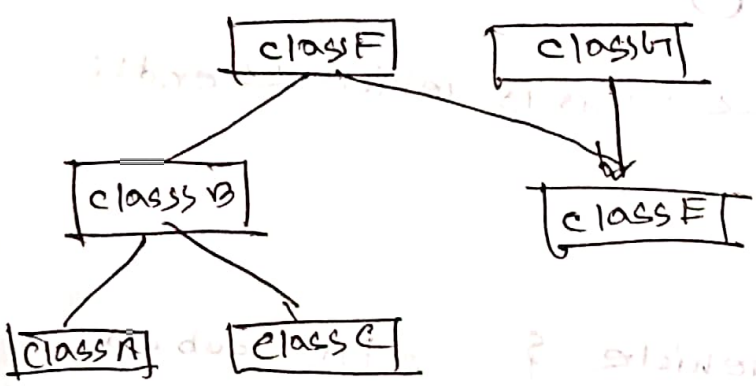
```
int main()
{
  car obj1;
  bus obj2;
  return 0;
}
```

Output

This is a vehicle

This is a vehicle

5. Hybrid (virtual) Inheritance : Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.




```
#include<iostream>
using namespace std;
class vehicle {          // base class
    public:
        vehicle() { cout<<"This is a vehicle"<<endl;
        }
};
```

```
class Fame {      // base class
    public:
        Fame() { cout<<" Fame of vehicle"<<endl; }
    };
class car: public vehicle {    // First sub-class
        };
class bus : public vehicle, public Fame {
    };
int main ()
  {
  bus obj2;
  return 0;
  }
```

Output

This is a vehicle

Fame of vehicle

☐ What are the differences between single and multiple inheritance in c++ language?

Single Inheritance	Multiple inheritance
It is the one where in the derived class inherits the single base class	The derived class derives from two or more base classes.
The derived class inherits the features of the base class.	The derived class uses the combined features of the multiple base classes.
It requires a small num time in comparison to multiple inheritance.	It requires more num time in comparison to single inheritance
It has less overhead	It has more overhead
It is a generalization.	It is a specialization
It is a simple in comparison to multiple inheritance	It is complex ~~to~~ in comparison to single inheritance.
It can be implemented in any programming language	Only c++ supports multiple inheritance

SIMRIF

Chapter-9

1. What does polymorphism mean in C++ language?

The word "polymorphism" means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

A real life example of polymorphism is a person who at the same time can have different characteristics. A man at the same time is a father, a husband and an employee. So the same person exhibits different behaviour in different situations. This is called polymorphism.

```
#include <iostream>
using namespace std;
class Geeks {
  public:
    void func(int x)
      {
       cout << " Value of x is " << x << endl;
      }
    void func(double x)
      {
       cout << " Value of x is " << x << endl;
      }
    void func(int x, int y)
      {
       cout << "Value of x and y is " << x << ", " << y << endl;
      }
};
```

```
int main()
  {
   Greeks obj1;
    obj1.func(7);
    obj1.func(9.132);
    obj1.func(85, 64);
    return 0;
  }
```

Output:

Value of x is 7.
Value of x is 9.132
Value of x and y is 85, 64

2. What is a virtual function ? What are the rules for virtual functions ?

A virtual function is a member function which is declared within a base-class and is re-defined by a derived class. When we refer to a derived class object using a pointer or a reference to the base class, we can call a virtual function for that object and execute the derived class's version of the function.

Rules for Virtual Function:

1. The virtual functions must be members of some class.
2. They cannot be static members.
3. They are accessed by using object pointers.
4. A virtual function can be a friend of another class.
5. A virtual function in a base class must be defined, even though it may not be used.
6. The prototypes of the base class version of a virtual function and all the derived class versions must be identical.
7. We cannot have virtual constructors, but we can have virtual destructors.
8. If a virtual function is defined in the base class, it need not be necessarily redefined in the derived class. In such cases, calls will invoke the base function.

SHARIF

Java

Class, object and method, package

1. Explain class, object and method in java language.

Class: A class is a blueprint in the java programming language from which an individual object can be built. In java, we may declare a class by using the class keyword. Class members and functions are declared simply within this class

syntax to declare a class:

```
access-modifier class class-name
{
data member;
method;
constructor;
nested class;
interface;
}
```

Objects: Once the class type has been defined, we can create "variables" of that type using declarations that are similar to the basic type declarations. In Java, these variables are termed as instances of classes, which are actual objects.

As pointed eanlier, an object in java is essentially a block of memory that contains space to store all the instances variables.

Objects in Java are created using the new operator. The new operator creates an object of the specified class and returns a refference to that object.

Example of creating an object of type Rectangle.

```
Rectangle rect1;  // declare the object
rect1 = new Rectangle();  // instantiate the object.
```

Method: Method in Java or Java Method is a collection of statements that perform some specific task and return the result to the callen. A java method can perform some specific task without returning anything. Methods in java allow us to reuse the code without retyping the code.

Syntax: Declare a method

```
<access_modifier> <return_type> <method name> (list
                                    of parametems)
{
  // body
}
```

Method declaration

Method declaration has six components.

1. Modifier: It defines the access type of the method such as public protected, private and defalt.

2. The return type: The data type of the value returned by the the method or void if does not return a value.

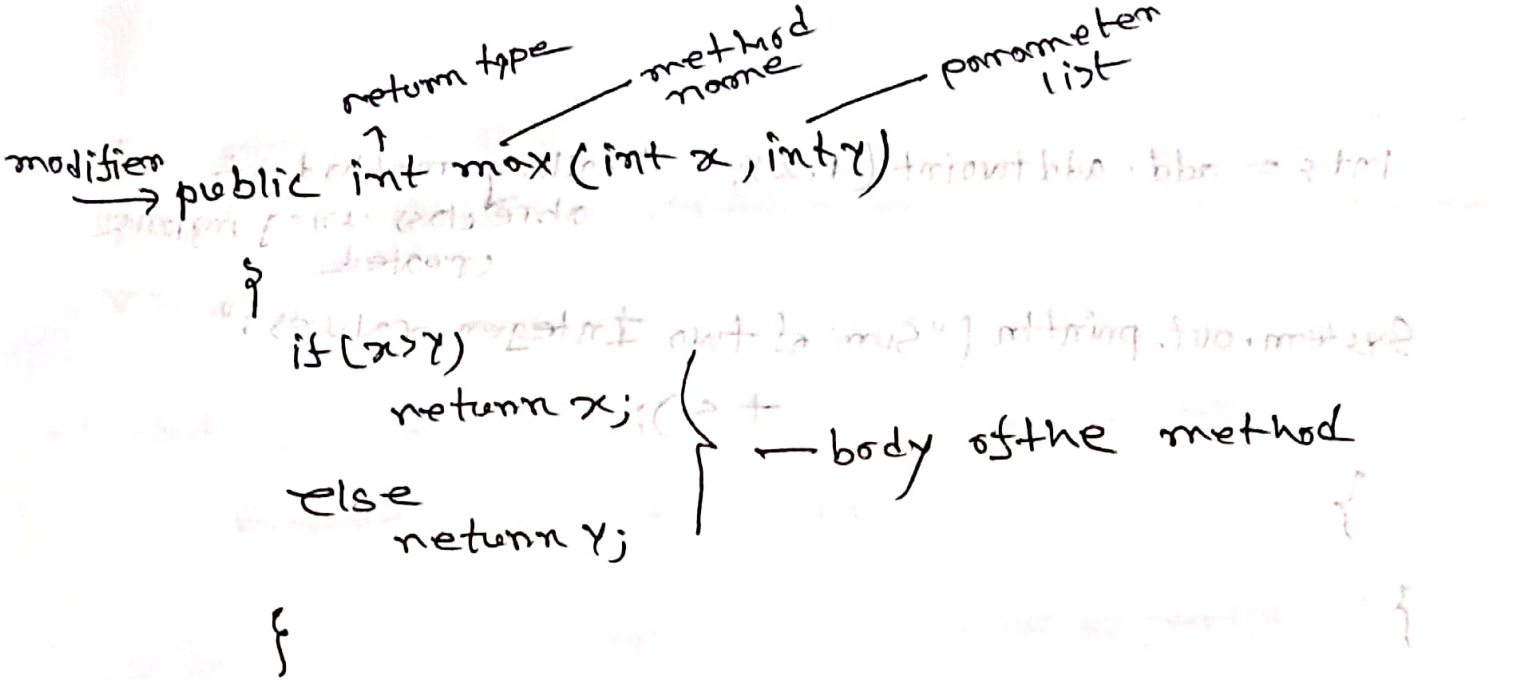
3. Method name: the rules for field names apply to method names.

4. Parameter list: Comma separated list of the input parameters is defined, preceded with their data type, within the enclosed parenthesis.

5. Exception list: The exception you expect by the method can ~~through~~ throw, you can specify these exception (s).

6. method body: It is enclose between braces.

The code you need to perform your intended operation.



Example:

```
import java.io.*; // importing required class

class addition {     // class-1, helper class

  int sum=0;
  public int addtwoint (int a, int b)  // method
   {

    sum= a+b;
    return sum;
   }
}

class GFG {      // class-2, Helper class

   public static void main (String[] args)  // main driver method
   {
    addition add = new addition();  // creating object of class 1 in main() method
```

```
int s = add.addtwoint(1,2);   // calling method of above class using instance created

System.out.println("Sum of two Integer values:"
                                    + s);
    }
}
```

Output

Sum of two integer values: 3

☐ How to declare field in Java?

Data is encapsulated in a class by placing data fields inside the body of the ~~function~~ class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated. We can declare the instance variables exactly the same way as we declare local variables.

Example

```
class Rectangle
{
    int length;
    int width;
}
```

The class Rectangle contains two integer type instance variables. Instance variable are also known as member variables.

☐ Write a java program using class and object:

Example of method section.

☐ Define constructor. What are the special properties of constructor.

A constructor in java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling the constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new () keword, at least one contructor is called.

Special properties for java constructor:

1. Constructor name must be the same as its class name.
2. A constructor must have no explict return type
3. A java constructor cannot be abstract, static, final and synchronized.

```
class Greek
  {
  . - - - -
  - - - - ...
  Greek() {  // constructor
    {
  : - - - :
  -       -:
  }
```

Example:

```
class Rectangle
  {
  int length, width;
  Rectangle (int x, int y)   // Defining constructor
    {
    length = x;
    width = y;
    }
  int rectArea ()
    {
    return (length * width);
    }
}
```

```
class RectangleArea
 {
  public static void main (String args[])
   {
   Rectangle rect1 = new Rectangle (15,10);   //calling construc
   int area1 = rect1.rectArea();
   System.out.println ("Area1 = " + area1);
   }
 }
```

Output
Area = 150

☐ What do you mean by method overloading and method overriding in Java?

Method overloading: If a class has multiple methods having same name but different in parametens, it is known as Method Overloading. Different ways of method overloading in java such as changing the number of parametens, changing data types of the arguments, changing the order of the parameter of methods.

```
public class sum {
    public int sum (int x, int y) { return (x+y); }
    public int sum (int x, int y, int z)
        {
        return (x+y+z);
        }
```

```
public double sum (double x, double y)
    {
    return (x+y);
    }
public static void main(String args[])
    {
    sum s = new sum ();
    System.out.println (s.sum (10,20));
    System.out.println (s.sum(10,20,30));
    System.out.println (s.sum (10.5,20.5))
```

Output:

30
60
31.0

Method Overriding in Java:

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

* Method overriding is used for runtime polymorphism.

```
class Vehicle {
    void run ()
    {
        System.out.println ("Vehicle is running");
    }
}

class Bike extends Vehicle    // creating child class
{
public static void main (String args[])
{
    Bike obj = new Bike (); // object of child class
    obj.run ();
                // calling method
}
}
```

Output:

Vehicle is running

Difference between method overloading and overriding:

Method overloading	Method Overriding
Compile-time polymorphism	run-time polymorphism
It helps to increase the readability of the program	It is used to grant the specific implementation of the method which is already provided by its parent class or superclass
It occurs within the class	It is performed in two classes with inheritance relationship.
Method overloading may or may not require inheritance	method overriding always needs inheritance.
methods must have the same name and different signatures.	methods must have the same name and same signature.
the return type can or cannot be the same.	In method overriding, ~~methods~~ the return type must have the same ~~name.~~
Static binding is being used for overloaded methods.	Dynamic binding is being used for overriding methods.
poor performance due to compile time polymorphism	It gives better performance due to runtim polymorphism.
Argument list should be different while doing method overloading.	Argument list should be same in method orenidding.

☐ Define inheritance. Describe the multiple inheritance in Java?

The mechanism of deriving a new class from an old one is called inheritance. The old class is known as the base class or superclass or parent class and the new one is called the subclass or derived class or child class.

The syntax of Java Inheritance:

```
class subclass_name extends Superclass_name
{
  // methods and fields
}
```

The extends keyword indicates that you are making a new class what derives from an existing class.

Example:

```
class animal {
  void eat () { system.out.println ("Eating...."); }
  }

class dog extends animal {
      void bark () { system.out.println ("barking...."); }

      }

Class TestInheritance {
   public static void main (String args[]) {
   dog d = new dog();
   d.bark();
   d.eat();
```

```
    }
}
```

Output

banking-----

eating.---

9. What is an interface? What is the major difference between interface and a class?

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, interfaces can have abstract methods and variables. It cannot have a method body.

syntax:

```
interface <interface_name>
    {
    // declare constant fields
    // declare methods that abstract
    {
```

```
import java.io.*;
interface Int {
    final int a=10;  // public, static, final
    void display();  // public and abstract
}
class Testclass implements Int {
    public void display() {      // Implementing the capabilities
    System.out.println("Geek");  of interface
    }
public static void main (String[] args)
    }
    Testclass t = new Testclass();
    t.display();
    System.out.println(a);
    }
{
```

Output
Geek
10

Difference between interface and class:

class	Interface
A class can have both an abstract and concrete methods.	Interface can have only abstract methods.
Multiple inheritance is not supported.	Interface supports multiple Inheritance.
final, non-final, static and non-static variables supported.	Only static and final variables are permitted.
A class can implement an interface.	An interface cannot implement an interface.
A class is declared using class keyword	Interface is declared using interface keyword.
A class can inherit other class	Interface can inherit only an interface.
A class can have any type of members like private public	Interface can only be have ~~implemented~~ public members
A class can have constructor	Interface can not have a constructor.

10. What is java packages? Write down the application of java package.

A java package is a group of similar types of classes, interfaces and sub packages.

package in java can be categorized in two form, built in package and user-defined package.

There are many built in package such as java, lang, awt, javax, swing, net, io, util, sql etc.

Applications of java package:

1. Preventing naming conflicts: For example there can be two classes with name Employee in two packages, college.staff.cse.employee and college.staff.ee.employee.

2. Making searching / Locating and usage of classes, interfaces, enumerations and annotations easier.

3. Providing controlled access : protected and default have package level access control. A protected member is accessible by classes in the same package and its sub-classes. A default member is accessible by classes in the same package only.

4. Packages can be considered as data encapsulation.

11 How you add a class or interface to a package? Explain

It is simple to add a class to an existing package.

Consider the following package:

```
package P1;
public class A
{
 // body of A
}
```

The package P1 contains one public class by name A. Suppose we want to add another class B to this package. This can be done as follows:

1. Define the class and make it public
2. Place the package statement

```
package P1;
```

before the class definition as follows:

```
package P1
public class B
{
 // body of B
}
```

3. Store this as B.java file under the directory P1.
4. Compile B.java file. This will create a B class file and place it in the directory P1.

Now the package P1 will contain both the classes A and B.

A statement like

```
import p1.*;
```

How do you create a package and design a package in Java?

For creating a package

(i) Declare the package at the biginning of a file using the form

```
package packagename;
```

(ii) Define the class that is to be put in the package and declare it public.

(iii) Create a sub-directory under the directory where the main source files are stored.

(iv) Store the listing as the classname.java file in the subdirectory created.

(v) Compile the file. This creates .class file in the sub-directory.

Example:

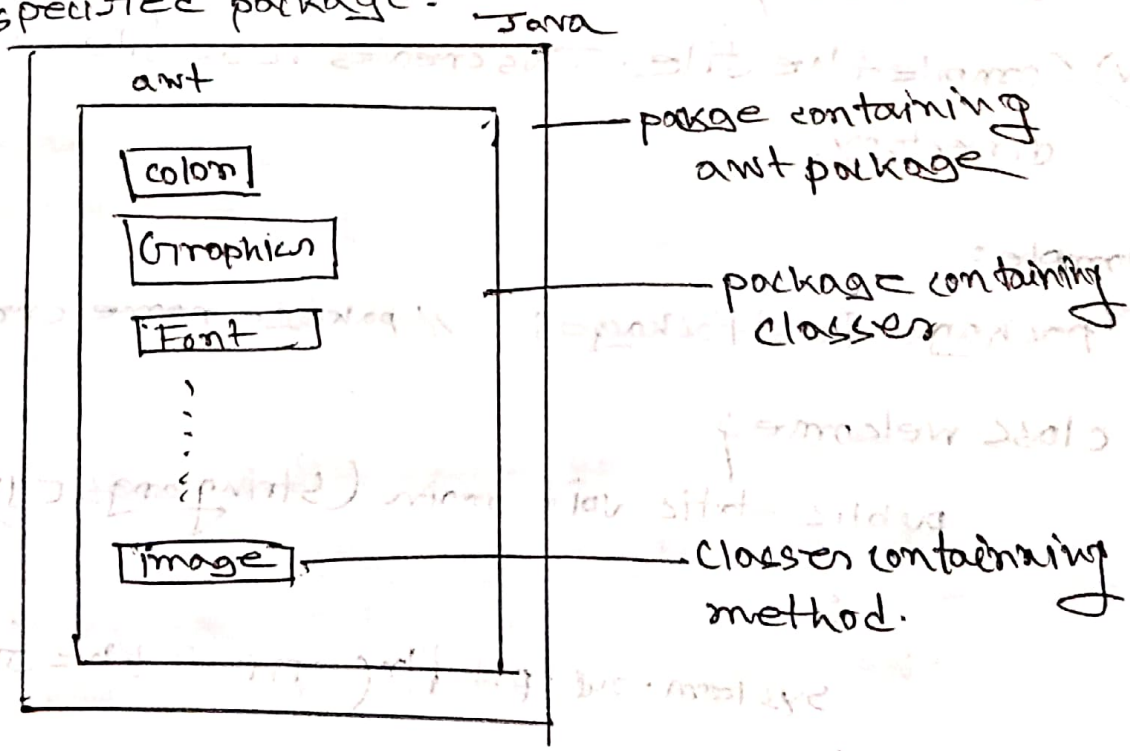
```
package FirstPackage;    // pakage name created
class welcome{
    public static void main (String args[])
        {
        System.out.println("This is the first program");
        }
}
```

The packages are organised in a hierarchical structure. This shows that the package named java contains the package awt, which in turn contains various classes required for implementing graphical user interface.

There are two ways to access in the classes stored in a package. The first approach is to use the fully qualified class name of the class that we want to use. This is done by using the package name containing the class and then appending the class name to it using the dot operator. If we want to refer to the class Color in the awt package

```
java.awt.Color
```

The second statement imports every class contained in the specified package.



Hierarchical representation of java.awt package

SHARIF

Exception Handling

1. What is an Error and Exception ? Describe their types with example.

Error: Errors are problem that mainly occur due to the lack of system resources. It cannot be caught or handled. It indicates a serious problem. It occurs at run time. These are always uncheaked.

Types of error:

Compile time error: All syntax errors will be detected and displayed by the Java compiler and therefore these errors are known as compile time error.

Example:

```
class error1
  {
   public static void main (String args[])
      {
       System.out.println ("Hello Java!")  // missing(;)
      {
  }
```

Here semecolon is missing.

Run-time error:

Sometimes, a program may compile sucessfully creating the class file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow.

Example

(1) Dividing an integer by zero.

(2) Accessing an element that is out of the bounds of an array.

(3) converting invalid string to a number etc.

Example:

```
class error2
  {
  public static void main (String args[])
    {
     int a=10;
     int b=5;
     int c=5;
     int x= a/(b-c);     // Division by zero
    System.out.println("x=" +x);
    }
  }
```

Exception: The term exception is shorthand for the phrase exception event. It is an event that occurs during the execution of the program and interrupts the normal flow of program instructions. These are the errors that occur at compile time and run-time. It can be recovered by using the try-catch block and throws keyword.

E. (1) Find the problem (Hit the exception)
(2) Inform that an error has occured (Throw the exception)
(3) Reciere the error information (catch the exception)
(4) Take corrective actions (Handle the exception)

There are two types of exception:

(i) Cheaked exception: These exceptions are cleanly handled in the code itself with the help of try-catch blocks. Cheaked exceptions are extended from the java.lang.Exception class.

(ii) Uncheaked exception: These exceptions are not essential handled in the program code; instead the JVM handles such exceptions. Uncheaked exceptions are extended from the java.lang.RuntimeException class.

```
import java.util.Scanner;
public class ExceptionExample
{
public static void main(String args[])
{
Scanner sc = new Scanner (System.in);
System.out.println("Enter a number : ");
int number = sc.nextInt();
System.out.println ("You have entered :" +number);
}
}
```

2. Write some common JAVA Exception.

ArithmeticException → Caused by math errors such as division by zero

ArrayIndexOutOfBoundException → Caused by bad array indexes.

ArrayStoreException → Caused when a program tries to store the wrong type of data in an array.

FileNotFoundException → Caused by an attempt to access a non existent file.

IOException → caused by general I/O failures, such as inability to read from a file.

NullPointerException : caused by nefferencing a null object

NumberFormatException → caused when a conversion between strings and number fails.

OutOfMemoryException → caused when there is not enough memory to allocate a new object.

SecurityException → Caused when an applet tries to perform an action not allowed by the browser's security setting.

StackOverFlowException: caused when the system runs out of stack space

StringIndexOutOfBoundsException : caused when a program attemps to access a nonexistent character position in a string.

```
class Errors3
{
public static void main (String argsI[])
  {
    int a=10,
    inb b=5;
    int c=5;
    int x,y;
```

```
try
  {
   x=a\(b-c); // Exception here
  }
catch(ArithmeticException e)
  {
   System.out.println("Dirision by zero");
  }
  y=a\(b+c);
  System.out.println("y="+y);
  }
}
```

Output:

Division by zero
y=1

Difference between throw and throws in Java exception handling.

Throw	Throws
The throw keyword is used inside a function. It is used when it is required to throw an Exception logically.	The throws keyword is used in the function signature. It is used when the function has some statements that can lead to exceptions.
The throw keyword is used to throw an exception explicity. It can throw only one exception at a time	The throws keyword can be used to declare multiple exceptions, separated by a comma. Whichever exception occurs, it matched with the declared ones is thrown automatically then.
Syntax of throw now includes the instance of the Exception to be thrown. Syntax wise throw keyword is followed by the instance variable.	Syntax of throws keyword includes the class names of the Exceptions to be thrown. Syntax wise throws keyword is followed by exception class names.
Throw keyword cannot propagate cheaked exception. It is only used to propagate the uncheaked Exceptions that are not cheaked using the throws keyword.	Throws keyword is used to propagate the cheaked Exceptions only.

Applet programming

SHARIF

1. What is applet ?

Applets are small java programs that are primarily used in Internet computing. They can be transported over the Internet from one computer to another and run using the Applet Viewer or any Web browser that supports Java. An applet, like any application program, can do many things for us. It can perform arithmatic operations, display graphics, play sounds, accept user input, crete animation and play interactive games.

2. What is Local applet ?

An applet developed locally and stored in a local system is known as a local applet. When a Web page is trying to find a local applet, it does not need to use the internet and therefore the local system does not require the internet connection. It simply searches the directories in the local system and locates and loads the specified applet.

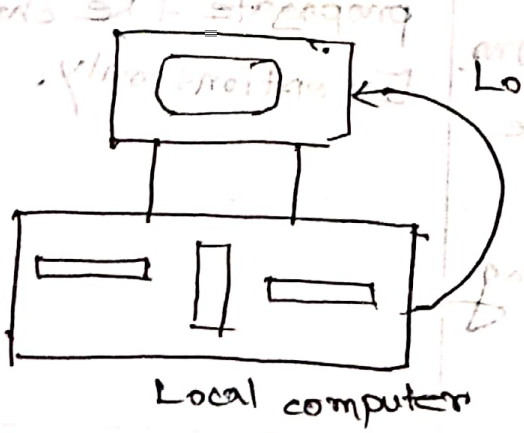


Figure: Loading local applets

3. What is remote Applets ?

A, remote applet is that which is developed by someone else and stored on a remote computer connected to the internet. If our system is connected to the internet, we can download the remote applet onto our system via at the internet and run it.

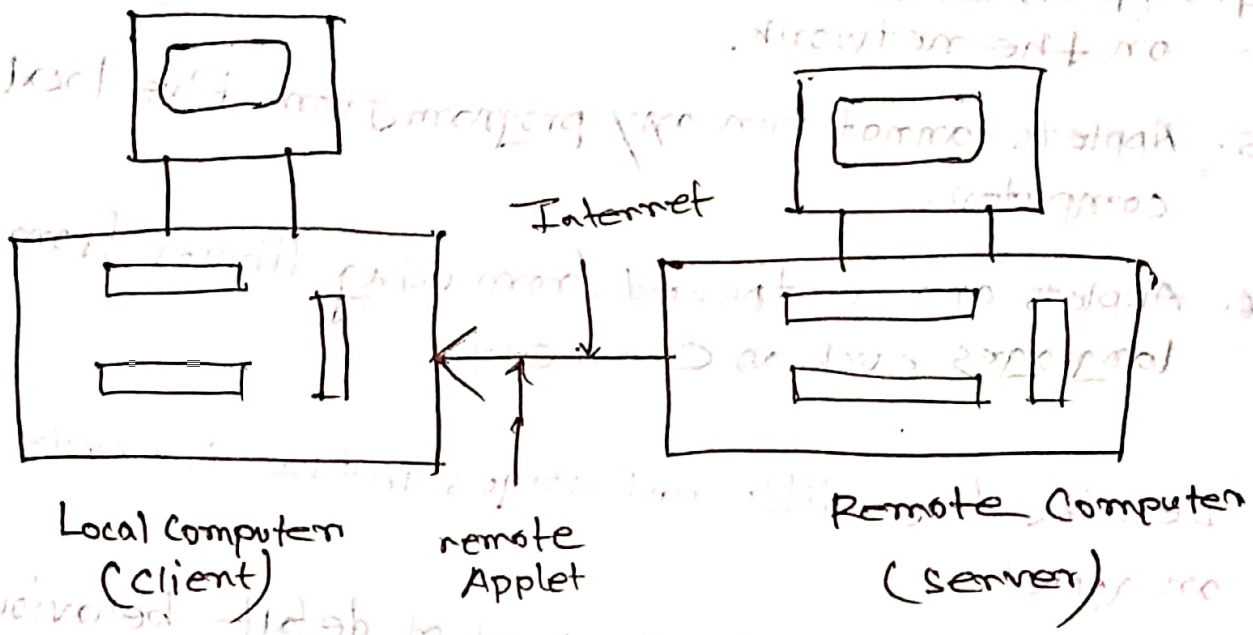


Fig: Loading a remote applet.

4. How do applets Differ from applications ?

1. Applets do not use the main() method for initiating the execution of the code. Applet, when loaded, automatically call certain methods of Applet class to start and execute the applet code.

2. Unlike stand-alone applications, applets cannot be run indendently. They are run from inside a web page using a special feature known as HTML tag.

3. Applets cannot read from or write to the files in the local computer.

4. Applets cannot communicate with other servers on the network.

5. Applets cannot run any program from the local computer.

6. Applets are restricked from using libries from other languages such as C or C++

□ Describe the different stages in the life cycle of an applet.

Every Java applet inherits a set of defalt behaviours from the Applet class. As a result, when an applet is loaded, it undergoes a series of changes in its state

The applet state include-

(1) Born or initialization state
(2) Running state
(3) Idle state
(4) Dead or destroyed state.

Initialization State: Applet enters the initialization state when it is first loaded. This is achieved by calling the init() method of Applet class. The applet is born. At this stage—

1. Create objects needed by the applet
2. Set up Initial values
3. Load images on fonts.
4. Set up colors.

To provide any of the behaviours mentioned above, we must override the init() method:

```
public void intit()
  {
  - - - -
  - - - -  (Action)
  }
```

Running State: Applet enters the running state when the system calls the start() method of Applet class. This occurs automatically after the applet is initialized. Starting also can occur if the applet is already in "stopped (Idle) state. For example, we may leave the Web page containing the applet temporarily to another page and return back to the page. This again starts the applet running. The start() method may be called more than once.

We may override the start() method to create a thread to control the applet.

```
public void start()
  {
  - - - - .. (Action)
  }
```

Idle or Stopped State: An applet becomes idle when it is stopped from running. Stopping occurs automatically when we leave the page containing the currently applet. We can also do so by calling the stop() method explicitly.

```
public void stop ()
  {
  . ——
  ——— (Action)
  — — —
  }
```

Dead State: An applet is said to be dead when it is removed from memory. This occurs automatically by invoking the destroy() method when we quit the browser. Like initialization, destroying stage occurs only once in the applet's life cycle.

```
public void destroy ().
  {
  — — —
  — — — (Action)
  — — —
  }
```

Display state: Applet moves to the display state whenever it has to perform some output operations on the screen. This happens immediately after the applet enters into the running state. The paint() method is called to accomplish this task. Almost every applet will have a paint() method.

```
public void paint (Graphics g)
{
    ------> (Display statements)
}
```

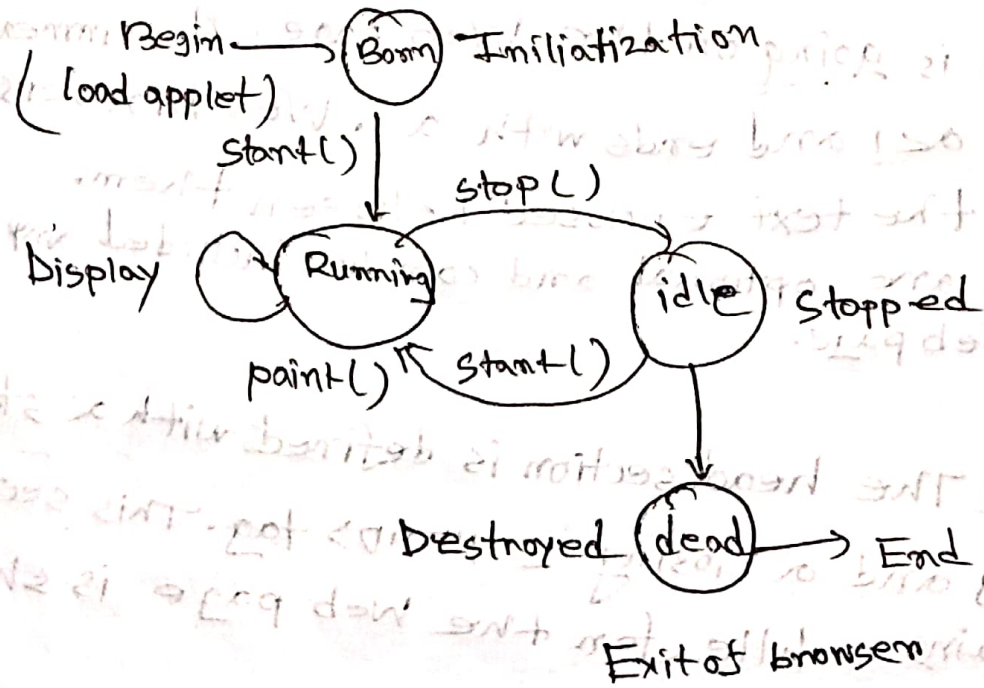


Fig: An applet's state transition diagram.

Describe the various sections of Webpage.

A webpage is basically made up of text and HTML tags that can be interpreted by a web browser or an applet viewer. A web page is also known as HTML page or HTML document.

A web page is marked by an opening HTML tag <HTML> and a closing HTML tag </HTML> and is divided into the following three major sections:

1. Comment Section (Optional)
2. Head section (optional)
3. Body section

Comment Section: This section contains comments about the Web page. It is important to include comments that tell us what is going on the Web page. A comment line begins with a <! and ends with a >. Web browsers will ignore the text enclosed between them. Comments are optional and can be included anywhere in the Web page.

Head Section: The head section is defined with a starting <HEAD> tag and a closing </HEAD> tag. This section usually contains a title for the Web page is shown below:

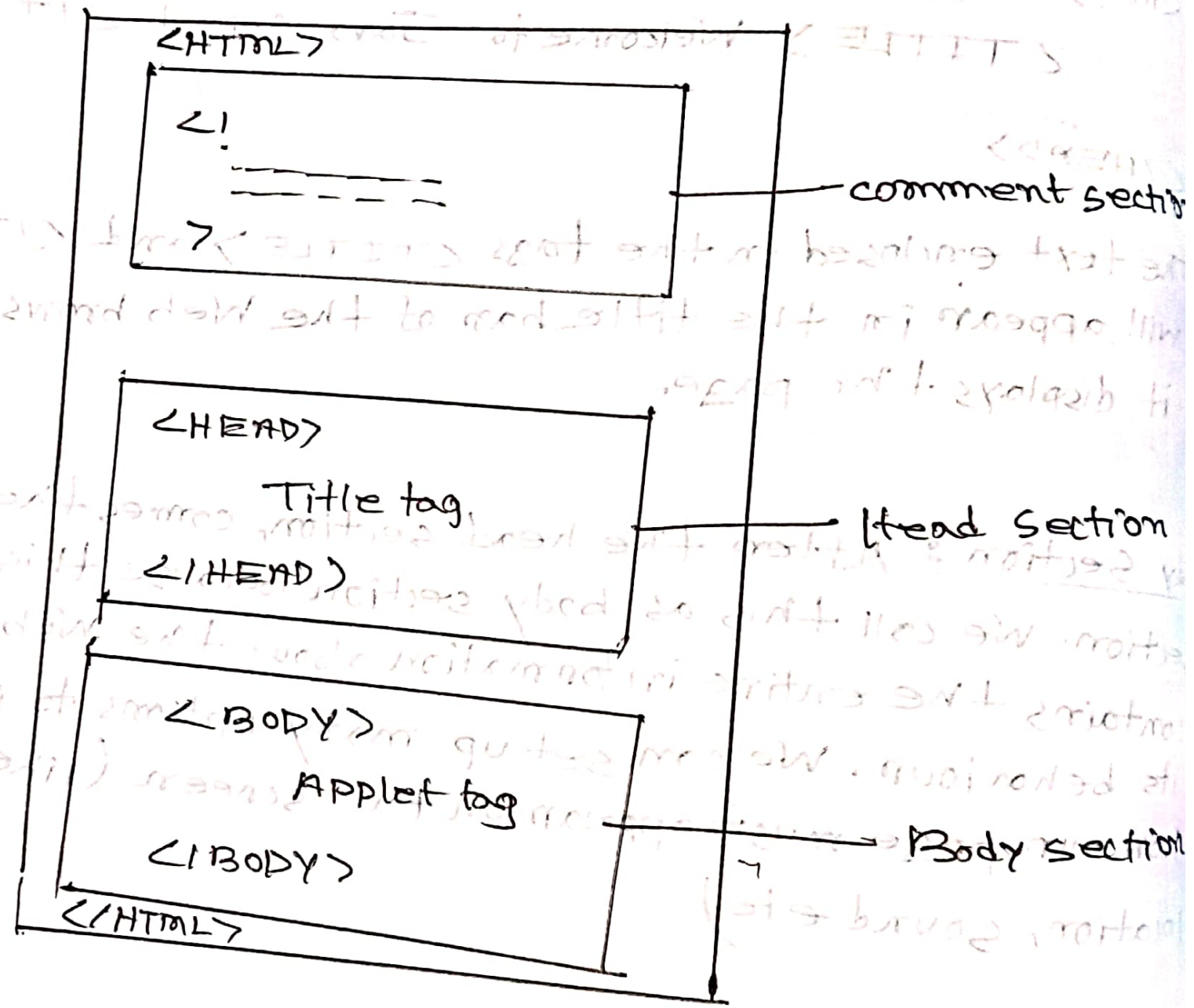
```
<HEAD>
    <TITLE> Welcome to Java Applet </TITLE>
</HEAD>
```

The text enclosed in the tags <TITLE> and </TITLE> will appear in the title bar of the Web browser when it displays the page.

Body Section : After the head section, comes the body section. We call this as body section because this section contains the entire information about the Web page and its behaviour. We can set up many options to indicate how our page must appear on the screen (like color, location, sound etc)

```
<BODY>
    <CENTER>
        <H1> Welcome to the world of Applets </H1>
    </CENTER>
    <BR>
    <APPLET-->
    </APPLET>

</BODY>
```


<HTML>
<!

>
comment section
<HEAD>
Title tag.
</HEAD>
Head section
<BODY>
Applet tag
</BODY>
Body section
</HTML>