Date:

Quiz #1

OOP vs Procedural prog...

OOP	Procedural pro...
In OOP, the program is broken into tiny components called object.	In procedural programming the program is broken into components known as functions.
Object oriented programming follows bottom up approach.	Procedural programming follows top-down approach.
OOP has access specifiers such as private, public, protected etc.	In procedural programming there is no access specifer.
Adding new data and functionality is easy.	Adding new data and functionality is not easy.
~~Examples:~~ C, FORTRAN, Pascal, Basic etc.	~~Examples:~~ C++, Python, C# etc.

TWTFS

Date:

Class	Object
Combination of variables and functions	Objects are made from classes.
Class is a logical representation of data	Object is a physical representation of data.
When we create class it does not allocates memory.	When we create object, it allocates memory.
We can create class with the help of class keyword	We can create object, with the help of new keyword.
Syntax : class Classmate	Class_name obj_name = new Class_name();

Data member	Member function
Data members are the data variables.	Member functions used to manipulate these variables together.
~~The~~ or In data members, we can store any value.	Member functions access the values of the data members and perform operations on the data members.
Data members can also be:	It can be defined inside or outside the class.
* Objects	
* Array of objects	
* Pointers to objects.	

Date:

M T W T F S

Public access specifier

Members are accessible from outside the class. It can be inherited.

Private access specifier

Members cannot be accessed from outside the class. It cannot be inherited.

Constructor:

It is a function which has same name as class. It has no return type even that we will not write void. They may have parameters or may not have.

Types:

Arguments

Without argument

Purpose:

It is used to initialize data members of an object i.e it gives initial value

NOTES

Date:

Syntax:

```
public class Myclass
{
    Myclass( ) → This is a constructor
    {
    }
```

How to create an object:

We can create an object by using a "new" keyword. This is the most common and basic way of creating an object in java. In this method we can call constructors with parameters or with no arguments. The new keyword allows us to create a new object.

Syntax:

```
class-name object_name = new class-name( );
```

Role of "new" keyword:

It creates a java object and allocates memory for it. It is also used for array creations. as arrays are also objects.

M T W T F S

Date:

How to access properties and function of object:

In OOP, we can access properties (attributes) and functions (methods) of object by using dot notation.

Accessing properties

To access the properties, you simply use the dot notation/operator. followed by the name of the field/object.

```
myclass Myobject = new Myclass();
int value = myobject.myfield;
```

Accessing Functions

To call a function of an object, you can also use dot operator, followed by the method name, and any required arguments within parenthesis.

```
Myclass myobject = new Myclass();
myobject.mymethod();
int result = myObject.calculate Sum(3,3);
```

Define properties and behavior of object:

Properties are deata with names; while, behaviours are the actions, an object can perform on its properties. A java class without properties and behaviors is generally useless.

NOTES

Date:

```
public class person
{
    private String name;
    private int age;
    // constructor to initialize
    public person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
    // set the name
    public void setName(String name)
    {
        this.name = name;
    }
    // set the age
    public void setAge(int age)
    {
        this.age = age;
    }
    // get the name.
    public String getName()
```

```
Public Person()
{
    this.name = Unknown;
    this.age = 0;
}
```

```
class Person
{
    private String name;
    private int age;

    Person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

    Person()
    {
        this.name = "Unknown";
        this.age = 0;
    }

    void setName(String name)
    {
        this.name = name;
    }

    void getAge(int age)
    {
        this.age = age;
    }
```

Date:

Date: M T W T F S

```
String getName()
{
    return name;
}
int getAge()
{
    return age;
}

Class Suman;
{
public Static void main(String[] args)
{
Person person1 = new Person();
person1.setName("John");
person1.setAge(25);
System.out.println("Name: " + person1.getName());

S.O.P("Age: " + person2.getAge());
Person person2 = new Person();
person2.setName("Alice");
person2.setAge(30);
```

Access specifiers:

Protected:

The protected access specifiers is specified using the keyword "protected" The methods or data members declared as protected are accessible within the same package or subclasses in different packages. The access level of protected modifier is within the package and outside the package through child class. If you don't make the child class, it cannot be accessed from outside the package.

Default:

The access level of a default modifier is only within the package. It is used when no access specifier is specified for a class, method, or variable. It is also known as package-private because it restricts access to only within the same package.

Array of objects:

An array of objects in java is simply an array where each element is an object rather than a primitive data type. In Java, you can create an array to hold objects of a particular class. For example, if you have a class called "Myclass", you can create an array of objects of that class like this:

```
MyClass[] myArray = new MyClass[5];
```

This can create an array of MyClass objects with length of 5.

Object as function parameter:

In Java, you can pass objects as parameters to function just like you pass other data types. When you pass an object as a parameter to a function, you are actually passing a reference to the object, not the object itself.

We can pass object as function parameters by using object.clone() method or define a constructor that takes an object of its class as a parameter.

Object as data members:

We can make data members of other class from object of a class. This allows to create more complex data structures by composing objects within other objects. It is used to represent attributes or properties of that class.

```
class Address
{
    String street;
    public Address (String street)
    {
        this.street = street;
    }
class Person
{
    String name;
    Address address //object of Adress class
    public Person (String name, Address address)
    {
        this.name = name;
        this.address = address;
    }
}
```

```
public class main {
    public static void main(String[] args) {
        Address personAddress = new Address("Abc");
        Person person = new Person("John", "Abc");
        S.O.P ("Name" + person.name);
        S.O.P ("Address" + person.address);
    }
}
```

Inheritence:

Inheritance in Java is a fundamental concept in object-oriented programming that allows a class to inherit properties and behaviors from another class (called super or base class). This relationship forms an "is a relationship", where subclass is a specialized" version of the superclass. It has reuse ability so it saves time.

For parent class:

For parent class, we can use these words Parent/Super/base, but in java we uses Super.

For child class:

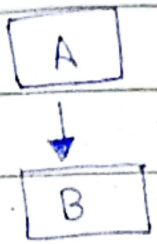
For child class, we can use these words child, Sub, derived but in java we uses Sub.

Types of inheritance:

i. Single Inheritance
ii. Multiple Inheritance
iii. Multilevel Inheritance
iv. Hierarchial Inheritance

Single Inheritance:

In single inheritance, a class can inherit from only one superclass. This is the default and most common type of inheritance in java.



Multiple Inheritance:

It's important to note that java does not support multiple inheritance for classes but it supports multiple inheritance through interfaces. A class can implement multiple inheritance/interfaces, allowing it to inherit the abstract methods and constants defined in those interfaces.

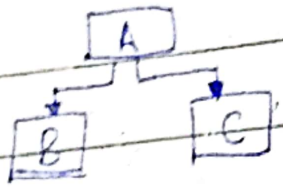
Multilevel inheritance:

In multilevel inheritance, a class inherits from another class, which in turn inherits from another class. This creates

a chain of inheritance.

Hierarchical Inheritance:

In hierarchical inheritance, multiple classes inherit from a single base class. This creates a tree-like structure of inheritance.



Polymorphism:

Polymorphism is a fundamental concept in OOP, and Java fully supports it. There are two main types of polymorphism in Java.

i Compile time Polymorphism (Method overloading)

ii Runtime Polymorphism (Method overriding)

Polymorphism in Java allows objects of different classes to be treated as objects of a common superclass.

Function overloading:

Function overloading in Java allows you to define multiple methods in a class with the same name but different parameters lists. The Java compiler distinguishes these methods based on the number or types of parameters and it's a form of compile-time polymorphism.

```
class Example {
    void display (int a) {
        S.O.P ("Method with 1 parameter: " +a);
    }
    void display (int a, int b)
    S.O.P ("Method with 2 parameters" +a+ "and"
            +b);
    }
    void display (String message)
    {
        S.O.P ("String message:" + message);
    }
}
```

They are overloaded because they have same name but different parameters.

Function Overriding:

Method / function overriding is a run-time polymorphism. It is performed in two classes with inheritance relationships. It always needs inheritance. It is used to grant the specific implementation of method which is already provided. The function must have same name and same signature. The return type must be same or co-variant.

```
class Animal
{
  void makeSound()
  {
    S.O.P ("Animal makes sound");
  }
}
class Dog extends Animal {
    void makeSound()
  {
      S.O.P ("Dog barks");
  }
}
class Cat extends Animal
```

```
void makeSound ( )
{
    S.O.P (" Cat meow");
  }
}
  Public      class Main
{
  Public static void main (String[ ] args)
{
    Animal a1 = new Dog( );
    Animal a2 = new Cat ( );
  }
}
```

"this" vs super keyword:

The "this" keyword refers to the current object in a method or constructor. The most common use of the "this" keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

Super keyword:

The super keyword refers to the parent (class) objects. It is used to call supper class methods, and to access the supper class constructor. The most common use of the "super" keyword is to eleminate confusion between super classes and subclasses that have methods with the same name.

Non-Static data members	Static data members
Memory will be allocated each and every time whenever an object is created.	Memory will be allocated only one whenever the class is loaded in the main memory regardless of number of object creates.
Non-static data members store specific values.	Class variable store common values.
Non-static variable declaration never preceeded by a keyword static. They are also known as Object level data members.	Static type variable are always preceeded by "static keyword". These data members must be accessed

" with classname
" They are also known
" as class level data
" members.

Static Function	Non-Static Function
Static methods are used to perform single operating like opening the files, obtaining a DBMS connections etc.	Non-static functions are used to perform repetitive tasks.
Methods/functions definition must start with the static keyword.	Methods definition does not require a static keyword.
Each and every static function must be accessed within respective class name.	Each and every non-static function must be accessed with the respective object name.
Result of static function are always shared by objects of same class.	Result of non-static function is not shared. Each object has its own copy of non-static method/function.

Abstraction:

Abstraction in Java is a process of hiding the implementation details from the user and showing only the functionality to the user. It can be acheived by using abstract class methods, and interfaces. An abstract class is a class that cannot be instantiated on its own and is meant to be inherited by concrete classes.

Abstract Class	Concrete class
A class declared with an abstract keyword which is a collection of abstract and non-abstract methods.	A class that allows to create an instance or an object using "new" keyword is concrete class.
Programmers cannot create objects using abstract class.	Programmer can create object using Concrete class.
An abstract class can have unimplemented method.	All methods in concrete class are implemented.

Interface:

An interface is a completely "abstract class" that is used to group related methods with empty bodies. To access the interface must be implemented with the "implements" keyword. Interface cannot be used to create objects. Interface methods do not have bodies and it can't contain constructors. On implementation of an interface you must override all of its methods. Java does not support multiple inheritance. However it can be acheived with interfaces, because class can implement multiple interfaces.

Has/Is a relationship:

In Java, we have 2 types of relationships

i Is a relationship

ii Has a relationship

i Whenver one class inherits another class it is called an Is a relationship. It is wholly related to Inheritance. It can be simply acheived

by using "extends" keyword. It is used for code reuseability in Java and to avoid code redundancy. It is tightly coupled, which means changing one entity will affect another entity.

ii Has-a relationship:

Whenever an istance of one class is used in another class, it is called Has-A relationship. In Java, Has a relationship otherwise called composition. It is additionally utilized for code reusability in Java. It is a unidirectional association i.e a one-way relationship. In aggregation, both the entries can survive individually which means ending one entity will not affect the other entity.