Encapsulation

Access modifiers Packages

Packages :-

Packages in Java are used to group related classes which help in avoiding name conflicts

There are two types of packages

1. Built in packages
2. user defined packages

1 Built In packages

```
import java.util.*;
import java.Array;
import java.lang.String;
```

2 user defined packages

```
Package AnimalPackage;
Public class Animal {
                Public void makeSound() {
                        Syso ("Animal make sound");
                }
}
```

folder - Animal
file - Animal.java

```
Import AnimalPackage;
    Public class main {
            Public static void main (String args[]) {
Animal a1 = new Animal();
    a1.makeSound();
}}
```

output :-

Animal make sound

Constructor :-
Constructor is a block of code similar to the method. it is called when an instance of the class is created.

Rules / properties
- constructor must have same name as that of the class name.
- constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final & synchronized. but it can be private, public or protected
- constructor is called only for once as the object is defined

Types of constructor
1. non parameterized constructor
2. parameterized constructor
3. Copy constructor

1. Non Parameterized constructor

```
class student {
    string name ;
    int age ;

Public void printInfo() {
    syso (this. name);
    syso (this. age);

Student () {
    syso ("constructor is called");
}
}
main →
    Student s1 = new student();
    s1. name = "aman";
    s1. age = 24 ;
    s1. printInfo() ;
```

output :-
constructor is called
aman
24

Static Keyword :- The static keyword in java is mainly used for memory management. We can apply static keyword with variables, methods, blocks and nested classes

```
class Student {
    String name;
    static String school;

    public static void changeSchool () {
        school = "newSchool";
    }
}

main →

    Student.school = "JMV";  → static variable is directly accessed by class name
    Student s1 = new Student();
    s1.name = "tony";

    syso (s1.name);
    syso (s1.school);
```

output :-
tony
JMV

```
abstract class Animal {
    abstract void walk();  ← OR
}                          Public void walk() {
class Horse extends Animal {   }
    Public void walk() {
        syso ("walks on 4 legs");
    }
class chicken extends Animal {
    Public void walk() {
        syso ("walks on 2 legs");
    }
```

main →

```
Horse h1 = new Horse();
h1.walk();
```

output :-

walks on 4 legs

```
abstract class Animal {
    abstract void walk();
    Public void eat() {
        syso ("Animal eats");
    }
}
```

..... (same)

main →

```
Horse h1 = new Horse();
h1.walk();
h1.eat();
```

output :-

walks on 4 legs
Animal eats

Constructor overloading :-

If a class has more than one construc-tor with different parameters, so that every constructor can perform a different task

```
Class student {
  String name;
  int age;

  Student() {
    Syso("this is default constructor");
  }
  Student (String name, int age) {
    syso("this.name" + " " + this.age);
  }
}

main →
  Student s1 = new student();
  student s2 = new student("aman", 24);
```

output :-

this is default constructor

aman 24

Page No.
Date

Code of point 3 :-

```
class Animal {
    Animal () {
        Syso ("animal is created");
    }
}

class Dog extends Animal {
    Dog() {
        super();
        Syso ("Dog is created");
    }
}

main →
    Dog d1 = new Dog();
```

output :-
animal is created
dog is created

Method overloading (Polymorphism)

If a class has multiple methods having same name but different in parameters or different return types or different in number of parameter passed is known as method overloading

Rules also covered in definition

```
class Student {
  String name;
  int age;

Public void printInfo(String name) {
      Syso(name);
  }
Public void printInfo(int age) {
      Syso(age);
  }
Public void printInfo(String name, int age) {
      Syso(name + " " + age);
  }
}

main →
    Student s1 = new student();
      s1.name = "aman";
      s1.age = 24;

    s1.printInfo(s1.name);
    s1.printInfo(s1.age);
    s1.printInfo(s1.name, s1.age);
```

output :-

```
aman
24
aman 24
```

Multilevel Inheritance

```
class Animal {
    Public void eat() {
        syso ("eating");
    }}
class Animal extends Dog {
    Public void bark() {
        Syso ("barking");
    }}

class Animal extends Tiger {
    Public void run() {
        syso ("running");
    }}

main →
    Tiger t1 = new Tiger();
    t1.eat();
    t1.bark();
    t1.run();
```

output :-
eating
barking
running

3 Hierarchial inheritance

```
class Animal {
    Public void eat() {
        syso("eating");
    }
}
class Dog extends Animal {
    Public void bark() {
        syso("barking");
    }
}
class cat extends Animal {
    Public void meow() {
        syso("meowing");
    }
}
```

main →

```
cat c1 = new cat();
c1.eat();
c1.meow();
```

output :-

eating
meowing

Module - IV

1 Inheritance or Single level inheritance

```
class Animal {
Public void eat () {
    Syso (" eating ");
}}

class Dog extends Animal {
    Public void bark () {
        Syso ("barking");
    }}

Public class main {
    Public static void main (String args[]) {

Dog d1 = new Dog();
    d1. eat();
    d1. bark();
}
}
```

Output :-
eating
barking

What is Inheritance
It is a mechanism in which one object acquires all the properties & behaviour of a parent object

code of point -a

```
class Animal {
void eat() {
    syso ("eating");
}}
class Dog extends Animal {
    void eat() {
        syso("eating bread");
    }
    void bark() {
        syso ("barking");
    }
    void work() {
        super.eat();
        eat();
    }}
```

output :-
eating
eating bread

main →
Dog d1 = new Dog();
d1.work();

← NO

Rules :- methods should have same name
must have same parameter as parent method has
must include Inheritances

Method overriding

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

```
class Animal {
    void makeSound() {
        Syso("Animal makes sound");
    }
}

class Dog extends Animal {
    void makeSound() {
        Syso("dog barks");
    }
}

main →
Animal a1 = new Animal();
a1.makeSound();

Dog d1 = new Dog();
d1.makeSound();
```

output :-
Animal makes sound
dog barks

Multiple inheritance not supported in Java

```
class A {
    void msg() {
        syso("Hello");
    }
}

class B {
    void msg() {
        syso("welcome");
    }
}

class C extends A, B {          // Suppose
main {
    C c1 = new C();
    c1.msg();                   // Now which msg() method
                                   should be invoked
}
}
```

compile time error.

Page No.
Date

Super Keyword :-

The 'super' Keyword in Java is a reference variable which is used to refer immediate parent class object

Usage of super Keyword →

1. Super can be used to refer immediate parent class instance variable
2. Super can be used to invoke immediate parent class method
3. super() can be used to invoke immediate parent class constructor

--NO →

* Super is used before this Keyword & printing statement
* First parent class constructor is called then child class constructor

Code of point 1

```
class Animal {
String color = "white";
}
class Dog extends Animal {
String color = "black";
  void printColor() {
      syso (color); // prints color of dog
  } syso (super.color); // prints color of Animal
main →
  Dog d1 = new Dog();
    d1. printColor();
```

Output :-

white
black

Page No.
Date

Constructor chaining

```
abstract class Animal {
    Animal() {                          1st call
        syso("Creating a new Animal");
    }
}

class Horse extends Animal {
    Horse() {                           2nd call
        syso("Created a Horse");
    }
}

main →
    Horse h1 = new Horse();
```

Output :-

```
Creating a new Animal
Created a Horse
```

Note :-

It is important to write Public for base/derived class methods otherwise the access modifier will be default by default

→ Implements is used
→ Interfaces don't have constructors

Page No.
Date

Interface in Java

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

Properties

- All the fields (variables) in interfaces are public, static and final by default
- All methods are public & abstract by default
- imp A class that implements an interface must implement all the methods declared in the interface
- interfaces support the functionality of multiple inheritance

```
Interface Animal {
Public void walk(); //void walk(); → by default public & abstract
}
class Horse implements Animal {
    Public void walk() {                → implementation of method (abstr)
        Syso ("walks on 4 legs");
    }
}

main →
    Horse h1 = new Horse();
    h1.walk();
```

output :-
walks on 4 legs

Page No.
Date

Multiple inheritance using interface

```
interface Animal {
    void walk();
}
interface Herbivore {
    void eat();
}
class Horse implements Animal, Herbivore {
    Public void walk() {
        Syso("walks on 4 legs");
    }
    Public void eat() {
        Syso("eats Plants only");
    }
}

main →
    Horse h1 = new Horse();
    h1.walk();
    h1.eat();
```

output :-

walks on 4 legs
eats plants only

2 parameterized constructor

```
class Student {
    String name;
    int age;

    Public void PrintInfo() {
        Syso (this.name);
        Syso (this.age);
    }
Constructor → Student (String name, int age) {
        this.name = name;
        obj name   parameter
        this.age = age;
    }}

main →
    Student S1 = new Student ("Aman", 24);
    S1.printInfo();

output :-
    aman
    24
```

3 copy constructor :-

```
class Student {
    String name;
    int age;

    Public void PrintInfo() {
        Syso (this.name);
        Syso (this.age);
    }

    Student (Student s2) {
        this.name = s2.name;
        this.name = s2.name;
    }                    → Student() {
}                        }

main →
    Student S1 = new Student();
    S1.name = "aman";
    S1.age = 24;

    Student S2 = new Student (S1);
    S2.printInfo();

output :
    aman
    24
```

Page No.
Date

Abstraction :-

Abstraction is a process of hiding implementation details and showing only functionality to the user

Abstract class :-

A class which is declared as abstract (keyword) is known as an abstract class in Java

Properties

- An abstract class must be declared with an abstract keyword
- It can have both abstract and non abstract methods
- It cannot be instantiated // obj cannot be created of abstract class
- It can have constructors and static methods also
- It can have final methods which will force the subclass not to change the body of the method

```
Animal animal = new Animal();
    animal.walk();
```

Cannot instantiate the type animal
(Run time error)