Page No.
Date / /

OOPs

OOPS

Class → is a named group of properties and functions

→ it is template of an object
ex: cars, humans.

class → logical construct

Object → physical Reality // occupies space in memory.

object is an instance of a class

```
class Student{
    int roll;
    String name;
    float marks;
}
```

} instance variables

```
Student Sharath = new Student();
```

↳ by default constructor

it dynamically allocates memory at runtime

DOMS | Page No.
Date / /

→ constructor is a special function that runs when you create an object and assigns some variables

⇒ this key word used to substitute by reference variable.

Student Sharath = new Student(18, "Sharath");

Student (int rono, String naam){

this.name = naam;
this.rollno = rono;

if is substituted by sharath.

→ Constructor overloading.

⇒ final Keyword

→ for premitives like int if we use final keyword we cannot modify the value of variable

Ex: final int a = 10;

This a cannot be modified.
But final variable should always initialized

DOMS | Page No.
Date / /

→ for objects we can modify the data but we cannot reassign it

final Studen A = new Student("something")

A.name = "nothing"; . This works with final keyword but not reassigning other object.

→ in Java for premetives it's only pass by value. not pass by reference.
but for objects it pass by reference.

⇒ class
objects
new
constructor
final
Garbage collector

DOMS | Page No.
Date / /

⇒ OOPS 2

→ packages are nothing but folders

→ When a property is common to everyone (class) use Static.

```
Class Human {
    int age;
    int salary;
    Static long population
}
```

Here population is common property of the class

→ When you are access static variable always access with class name, you can also access through the reference variable of object but better way is use class name.

Ex: "Human.population" is right way than "this.population"

DOMS | Page No.
Date / /

Imp

→ Why Main is Static?

1) Main is static because it is first Method to run ~~&~~ when java program runs.
2) if it is not static it belongs to objects / instance.
3) Since Main is static, JVM starts the Java application, it needs a way to execute the code without requiring an instance of class. Declaring 'main' as static ~~main~~ that allow JVM to call directly using the class name without creating object of class.

→ Inner class and Singleton class

⇒ static vs non static → non predetermined
Static block
singleton class
inner class

Inner class

DOMS | Page No.
Date / /

⇒ OOPS3

Inheritance.

Class Box { child class can inherit from parent class.

double l
double w
double h.

}

Class BoxWeight extends Box {

double weight;

}

super key word calls the constructor of parent class

→ BoxWeight Box = new Box(. .);

this is not possible because, parent class doesn't know what are below classes

DOM 15 | Page No.
Date / /

→ Super Keyword explicitly refers to super Constructors or variables.

```
Box{
double weight;
double l;
}

Boxweight extends Box{
  double l;
  double w;
}
```

If you want to print or get 'l' from Boxweight class use this.l if you want to explicitly refer to Box class if use super.l.

→ final Keyword is used to avoid inheritance and overriding.

→ Static methods doesn't override.

polymorphism → setters and getters

→ Data hiding focus on data security
Encapsulation focus on hiding complexity

DOMS | Page No.
Date / /

⇒ Box {

}

BoxWeight extends Box {

}

→ BoxWeight bw = new Box();
→ Box bw = BoxWeight();

} find the difference between them

1st one is not possible mostly but it works after casting

&

poly → many

Morphy → ways to do / represent

Static / runtime polymorphism → Dynamic method dispatch (method overriding)

Dynamic / compile time polymorphism → Method overloading

DOMS | Page No.
Date / /

→ OOPS4

→ Access Modifiers table
→ imp rule for protected.
↳ it can be accessed in subclass in other packages.

→ OOPS5

Abstract class
↳ Methods should override.

Interfaces
↳ supports multiple inheritance.

→ OOPS6

Shallow copy → for primitive it creates new one but non primitives it points to same objects

Object comparing
Lambda functions
exception handling.
Object cloning.