



Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Full, Year:2024), B.Sc.in CSE (Day)

KSA 2 (Class Notes)
Course Title: Object Oriented Programming
Course Code: CSE201 Section:232-D4

Student Details

| Name | ID |
|----------------|-----------|
| 1. MD.SHAJALAL | 223002088 |

Submission Date : 11-12-2024
Course Teacher's Name : Md. Parvez Hossain

| <u>Project Status</u> | |
|------------------------------|-------------------------|
| Marks: | Signature: |
| Comments: | Date: |

Class Note Midterm

Note

Content

- o Overview of Java
- o Data types, Variables and Arrays.
- o Operators
- o Control Statements,

Overview of Java

What is Java?

An object Oriented Programming language developed at Sun Microsystems.

Java was conceived by James Gosling, Patrick Naughton, Chris North, Ed Frank, and Mike Sheridan at Sun Microsystems.

A set of Standardized class Libraries (Packages) that support:

i. Creating graphical user interfaces

ii. Communicating over networks

iii. Controlling multimedia data

Features of Java:

- Platform independent

- Object Oriented

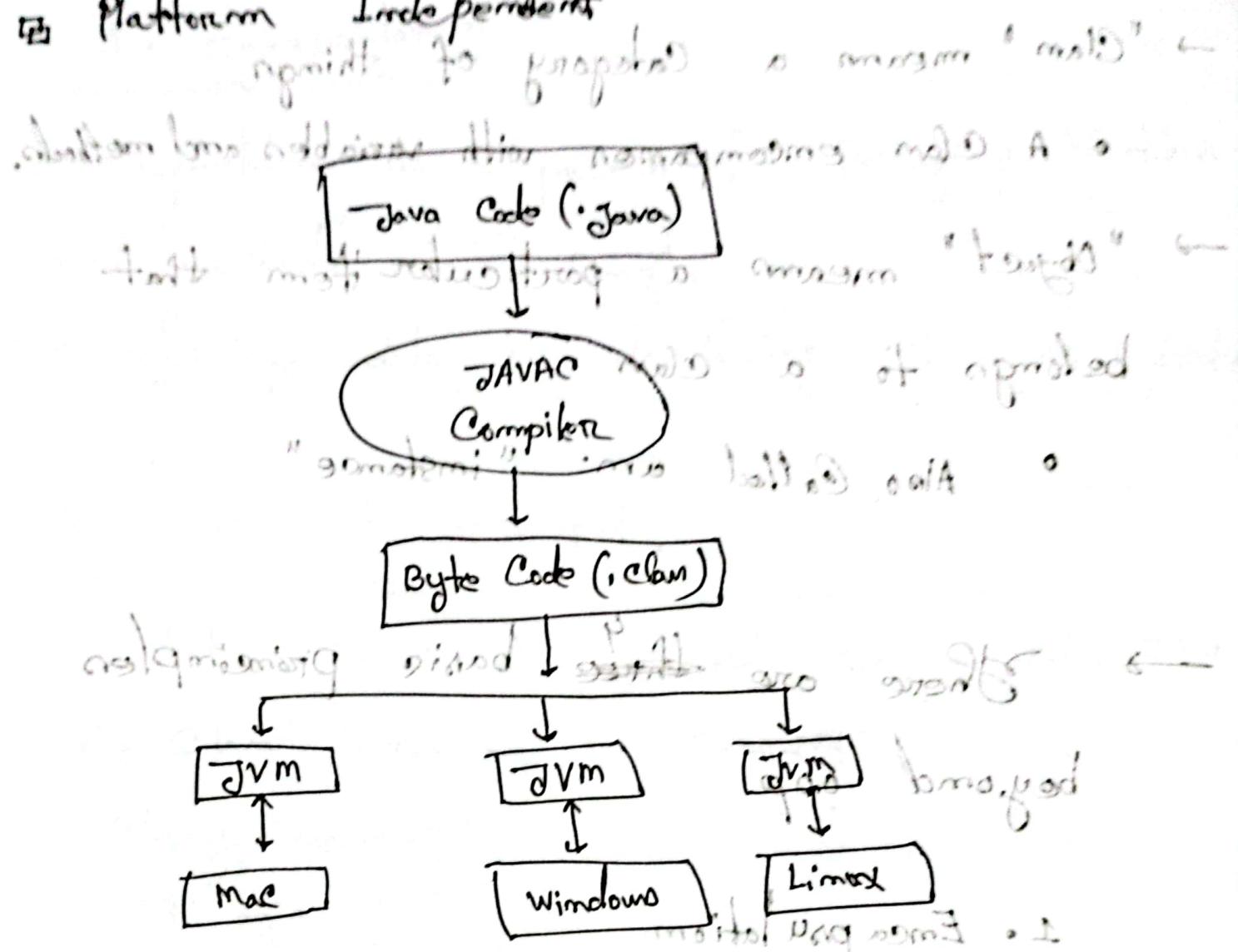
- Robust

- Secure

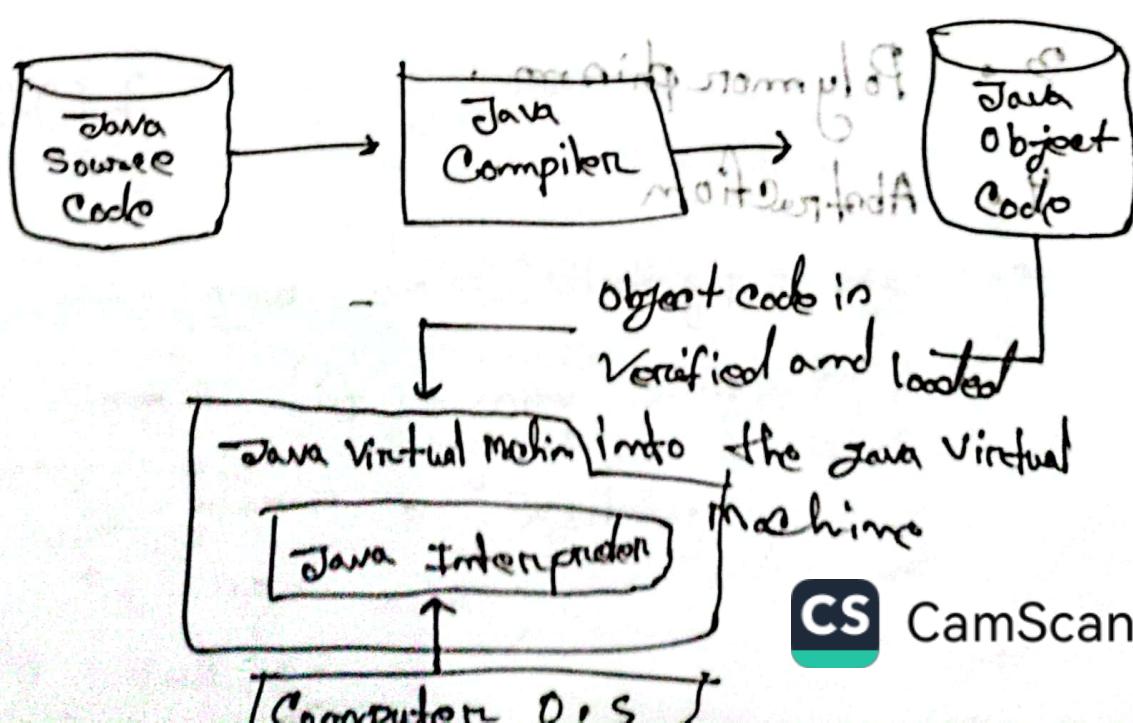
- Multi-threaded

{ note at footer

④ Platform Independent



④ Diagram of JVM



Object Oriented

→ "Class" means a category of things

- A Class encompasses with variables and methods.

(obj.) class obj.

→ "Object" means a particular item that belongs to a class

DATA

variables

- Also called an "instance"

(inst.) class obj.

→ There are ~~4~~ basic principles

beyond OOP

MVC

MVC

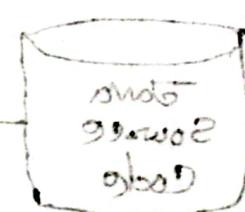
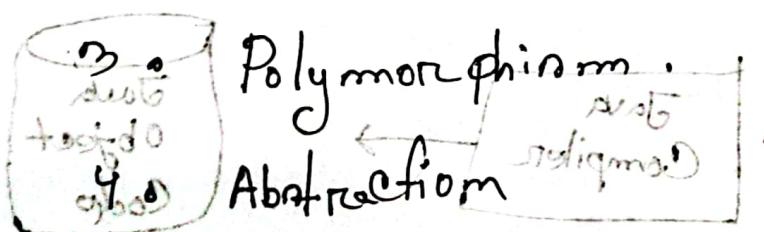
1. Encapsulation

private

public

2. Inheritance

MVC → morphism



abst & refl
abst & refl

Introducing ext. struc/inheritance and
composition

inheritance



CamScanner

→ Simple Java Program

package hello;

public class Hello

public static void main(String[] args)

System.out.println("Hello World")

main method must declare

Public : to get access to other Command
line interface instructions.

Static : to get access without creating
any instance.

Void : main do not return any value.

→ Data types, Variables and Arrays

Then Primitive Types

→ Java defines eight primitive types of data:
byte, short, int, long, char, float, double
and boolean.

→ There can be put in four groups:

• Integers : byte, short, int, long
8bit 16bit 32bit 64bit

• Floating point numbers : float, double.
32bit 64bit

• Characters : char.
16bit

• Boolean : boolean, value can be just true or false.

~~operator~~ ~~bitwise~~ ~~operator~~ ~~logical~~ ~~operator~~

→ Arithmetic Operators → Bitwise Operators → Logical Operators

| + | = | AND |
|----|---|-------------|
| * | = | OR |
| / | = | XOR |
| ++ | = | SHIFT right |
| -- | = | SHIFT left |

→ Assignment Operators

= → Logical operator → Relational

&& → AND & → OR
|| → OR

== → Equal to != → Not equal

! = → marked ! = → unmarked

→ Ternary

? :

~~Java Array~~

→ An array is a collection of variables of the same type.

Same type. $\{ \text{int} = [\text{of}] \text{medium} \}$

→ Using array

$\text{int}[] \text{number} = \text{new } \text{int}[10]$.

quint part

⇒ Declaring arrays → form ~~apostrophe~~

datatype[] arrayName ~~and~~ ~~slicing~~

~~(apostrophe [] points) mean below state slicing~~

Example

$\text{int}[] \text{number, medium } [] \text{for}$

$\text{String}[] \text{name, } [] = [\text{of}] \text{medium}$

⇒ Creating arrays

- For creating ~~and~~ an array you can use new operator $\text{op } = [\text{of}] \text{medium}$

$\text{int}[] \text{number} = \text{new } \text{int}[10]$

log two

or



CamScanner

→ Array Initialization

- int [] number = new int [10];

number[0] = 10; , ~~out put~~ ~~out put~~

number[1] = 20;

number[2] = 45;

* [0] ~~out put~~ ~~out put~~ == ~~number []~~

→ Array Example

```
package array - demo;
public class Array1 {
    public static void main (String [] args) {
        int [] number = new int [5];
    }
}
```

int [] number = new int [5];

number[0] = 10; ~~out put~~

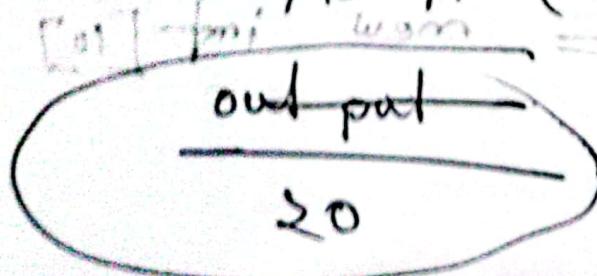
number[1] = 12; ~~out put~~

number[2] = 15; ~~out put~~

number[3] = 20; ~~out put~~

number[4] = 40; ~~out put~~

System.out.println (number[3]);



- char array {

param (string to write);

string buffer, sign = new int [5];

for (int i = 0; i < strlen(str); i++) {

System.out.println (sign[i]);

}

for (int i = 0; i < strlen(str); i++) {

if (str[i] != ' ') {

sign[i] = str[i];

No of non-space characters and sign.

Sign is to change

and sign is to change

{ non-space char }

odd even II

odd even II

⇒ Content

- Java Class { Class and Object }
- Java Objects.
- Example : Java Class and Objects
- Java methods (i.e. functions)
- Java constructors, methods

→ Java Class and Object

- A Class is a user defined blueprint or prototype from which objects are created.
- Java Class represents the set of properties or methods that are common to all objects of one type.

→ Java Class

```
class ClassName {  
    // variables  
    // methods
```

⇒ Java Object

(Apres l'initialisation) m = new Class-Name();

Class-Name Obj-Name = new Class-Name();

⇒ Java Methods

package Stud;

public class Student {

String name;

int id;

void Set (String n, String id, int i) {

name = s,

id = 2020;

void Pri() {

System.out.print("Name : " + name)

Methods

\Rightarrow Java Constructors

• Overloaded $\$ \sim$ in (String[], args)

Student $s_1 =$ new Student ("Khalid", 20)

$s_1 =$ primitive()

↳ constructor

? late object

? initialize and sizing

? own private

? bi

{ (int, boolean, String) to bio }

? $\varnothing = \text{empty}$

? $\varnothing = \text{bi}$

15

? (Ding bio)

String "empty" containing two methods

16

abstraction

\rightarrow sommerf (pd. No) & value pd. No. \leftarrow
~~et~~ \Rightarrow Content

■ Recursion

■ Call by value & Call by Reference.

■ Variable - Length Arguments.

eff mi value mit passing pd. bottom

→ Recursion \leftarrow , statements

public class Factorial {

• Recursive Call

 int fact(int m) {

 Base Case

 if m == 1, return 1

 else return m * fact(m-1)

 }

 public static void main (String [] args) {

 Factorial ob = new Factorial ()

 int result = ob . fact (5)

 System.out.println ("Factorial = " + result)

 }

y

→ Call by value & Call by Reference

Call by value

- In Java, call by value refers to calling a method by passing the value in the parameters.

- In Java, call by value is a copy of the variable to the method, so all changes are reflected only in that method; thus, no changes are reflected in the main method.

- When we pass a variable with primitive data types, it is considered a call by value, so any changes to the variable will not be reflected in the caller's scope.

Exam

public class CallByValue {

 main void change(int i) {

 i = 10; (i's value = 20 before) after swap

 }

 public class Test {

 public void m(String[] args) {

 CallByValue ob = new CallByValue();

 ob.value = 10; - pd - the value

 System.out.println("x before call : "+x);

 ob.change(x);

 System.out.println("x after call : "+x);

 } // main method

4

output - print

(int parameter) before call
10

"before" = same as 10



CamScanner

• Call by Reference and Silding

• If we call a method by passing - a - Re-type data (object, String etc.) then it is known in call - by - reference.

• Changes to that formal parameter does not affect the actual parameter.

• In Call - by - ref = x. original value gets changed.

(x+): Example) int main() {
 char name[10];
 cout << "Enter Name: ";
 cin >> name;
 cout << "Original Name: " << name;
 cout << endl;

char Reference {

String name;

void Change (Reference R2);

R2.name = "Rube";

{ }

Char - Test {

(psVnm) (Sub a) $\frac{f}{g}$
Reference r_0 = new Reference

Ref. name = $\text{emp}^n A \text{ Sogib}^{m+1}$

new $\text{S.O.P.}^n ("before\ full + ref. name")$

r_0 : change(r_0); obfuscation

S.O.P. ("after"; r_0 + r_1 , name)

y (new ... fin) also has
output
so new fin

f (new; x fin) Sogib

x + new = new

(new) with new, two on stage

so r_0 = new Sogib^{m+1}

(new EIP value) move to R9

Normal bits won't do much



CamScanner

Q7) Variable Length Arguments (Varargs)

→ Variable length arguments \rightarrow Varargs method
→ A method that takes a variable number of arguments is called a Varargs method.

Exam

```
public class AddDemo {
    void add (int ... num) {
        int sum = 0;
        for (int x : num) {
            sum = sum + x;
        }
        System.out.println (sum);
    }
}
```

public class Test

PS VM (String [] args)

AddDemo ob = new AddDemo();

ob.add (10, 20)

ob.add (10, 30, 40, 50)



CamScanner

1. Encapsulation

→ Encapsulation is a process of packaging variables

and methods into a single unit.

→ Protecting data by declaring them as private.

• Single Unit example:

Package oop;

public class Simple {

String name
int age

void eat() {

}
void talk() {

public static void main(String[] args) {

Simple s1

s1.name = "Sojib".

s1.age = 22;

s1.talk();



CamScanner

Encapsulation - ଭାବୀ ପରିଷ୍ଠି - ନିଯମ

4 Data Private

addition, private keyword to making o f information
Data Private \Rightarrow private key word ଶୁଦ୍ଧତା

Package Java;
public class Data {
 private String name;
 private int ID;
 void eat() {
 }
 void talk() {
 }
 }
 p s v m (String [] args) {
 }

private keyword \Leftrightarrow
- କୁଳ ମାତ୍ର ଏହାଙ୍କୁ

- ଅନ୍ୟଙ୍କୁ ଲାଗନ୍ତି ନାହିଁ
- ଶୁଦ୍ଧତା - ଏହାରେ ମାତ୍ରଣେ ନା
 କୋଣିବେଳୀ ଥିଲୁ

private \Rightarrow ଏହାକିମ୍ବଳୁ
- କୁଳରେ ଶୁଦ୍ଧତା କମିଛି
- ଥର୍ମ, method \Rightarrow କୁଳରେ
private keyword \Rightarrow ଶୁଦ୍ଧତା
- କୁଳରେ ଥିଲୁ

Data ($s_1 = \text{new Data}()$;)

s_1 : name = "Shayalal";
 s_1 : ID = 11012;

s_1 . talk();

bio \Rightarrow - ଜୀବିତ ରୂପ ଫର୍ମ
- କୁଳରେ କମିଛି

bio - ବାଧ୍ୟତା
Print method \Rightarrow ଶୁଦ୍ଧତା କମିଛି

get Name(), set "Name() ob method will be

→ so it's a private method.

Example p.c. has methods giving address, &

address with top line return of get

private String name;

private int id;

public String getName();

return name;

}

public void setName(String name);

this.name = name;

}

public static void main(String[] args);

Java N1 = new Java();

N1.setName("Shaghal");

System.out.println(N1.getName());

Y

→ How to do encapsulation? (Ques. Top)

1. Declare the variable ~~with~~ ^{private}.

2. Provide public ~~setters~~ and ~~getters~~ ^{method} ~~to~~ ^{with} ~~get~~ ^{set} the variables
value.

{ (Ques. Top) private value }

{ common method }

{ (common private) and (public value) }

{ common a non. value }

{ (after 2 private) } 2 9

{ (not use = & not use)

{ ("Intcode") private + &

{ (Ques. Top, &) intaking, two, making }

→ Exercise → 1 write a Java program to Create a class called Person with private int instance variables name, age, and country. Provide public getter and setter methods to access (and modify) these variables.

Class Person {
 private String name; // name = age, with CGE
 private int age; // age = age, with
 private String country; // country = age, with

Public String getName() {
 return name; } // name = age, with

return "name-age"; } // name = age, with

Public int getAge() {
 return age; } // age = age, with

return age; } // age = age, with

return age; } // age = age, with

Public String getCountry() {
 return country; } // country = age, with

return country; } // country = age, with

public void set Name (String name) {
 this.name = name; }

public void setAge (int age) {
 this.age = age; }

this.name = name; }
public void setCountry (String country) {
 this.country = country; }

public static void main (String[] args) {

Person s1 = new Person ("John", 22, "BD");

s2.set Name ("Shaghalan");

s2.setAge (22);

s1.setCountry ("BD");

String name = s2.get Name();

int age = s2.getAge();

String country = s1.getCountry();

System.out.println ("Name " + name);

→ Inheritance ← Inheritance

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance

1. Single Inheritance :
class Animal {
 void eat() {
 System.out.println("Animal is eating");
 }
}
class Dog extends Animal {
 void bark() {
 System.out.println("Dog is barking");
 }
}

Output :
Dog is barking
Animal is eating

class Dog extends Animal {
 void eat() {
 System.out.println("Dog is eating");
 }
}

Output :
Dog is eating

public class Main {
 public static void main (String [] args) {
 Dog n = new Dog();
 }
}

n.eat()
n.bark()

→ Multilevel Inheritance: Syntax

Class V {

 void start() {
 System.out.println("Vehicle");
 }

Class Car extends Vehicle {

 void drive() {
 System.out.println("Car");
 }

Class E extends Car {

 void ch() {
 System.out.println("Electric");
 }

(public class Main {
 public static void main(String[] args) {

 E s1 = new E();
 s1.start();

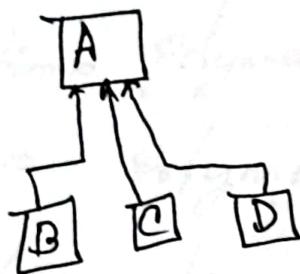
 s1.drive();

 s1.charge();

} (After Execution) Main class state will be

Vehicle
Car
Electric

→ Hierarchical Inheritance



Class A {
 public void met() {

System.out.println("method A");
 }

Class B extends A {

public void metB() {
 System.out.println("method B");
 }

Class C extends A {

public void metC() {
 System.out.println("method C");
 }

Class D extends A {

public void metD() {
 System.out.println("method D");
 }

public Class Test {

public static void main(String args[]) {

B N1 = new B();

C N2 = new C();

D N3 = new D();

N1.met();

N2.metB();

N3.metC();



2. Inheritance जटिलता

→ Inheritance is the process by which one object acquires the properties of another object.

→ Inheritance → जटिलता → extends keyword

→ जटिलता → वर्गीकरण संस्कृते के बारे में

public class Animal {

 int age; // उम्र
 String name; // नाम

public class Dog extends Animal {

 // बढ़ावा देता है

→ Why do we need inheritance?

- For code Reusability

- For method overriding

- To implement parent-child relationship

→ In Java, we use the extends keyword to inherit from a class.

Access Modifier

| | Class | package | Subclass | World |
|-----------|-------|---------|----------|-------|
| public | Yes | Yes | Yes | Yes |
| private | Yes | No | No | No |
| protected | Yes | No | Yes | No |

Java Method Overriding

→ Declaring a method in subclass which is already present in superclass is known as method overriding.

Java Method Overriding - ജാവ ഓഫൈസ്

→ സ്റ്റ്രോ കോംപാക്ട്

- Code reuse
- One interface, multiple implement
- Run time polymorphism.

Q what are the rules for Method overriding

blocks relative position no.

1. Name, Signature type, Parameter must be
no. of no. of no. of
Same.

2. If a method can't be inherited, then
it can't be overridden.

3. A method declared as final static
can't be overridden.

4. Comitrue-for Can't be overridden.

as class methods at bottom are finalised
as method of void requires in finalizing places

: finalise at bottom

using Pseudo - finalise at bottom first

(top) - original - bottom

sway go

and then algorithm, bootstrap goes

main program with me



CamScanner

Exam Method overriding

public class Person {

String name;

int age;

void displayInfo() {
System.out.println("Name : " + name);
System.out.println("Age : " + age);

}

→ method overriding

public class Teacher extends Person {

String qualification;

String name;

④ override

void displayInfo() {
System.out.println("Name : " + name);
System.out.println("Age : " + age);
System.out.println("Qualification : " + qualification);

System.out.println("Name : " + name);
System.out.println("Age : " + age);
System.out.println("Qualification : " + qualification);

System.out.println("Name : " + name);
System.out.println("Age : " + age);
System.out.println("Qualification : " + qualification);

Class Test

Person p = new Person();

Teacher t = new Teacher();

t.name = "Parvez";

t.age = 27;

t.qualification = "BSC in CSF";



s2.displayInformation() output

3 Name : Parvez Homai
4 output Name : Parvez Homai

Name : Parvez Homai Age : 27
Age : 27 Information about this person

Age : 27 Information about this person
Degree : BSC Jim CS

→ Same Question ←

Can we override static methods?

Ans : No, static method can't be overridden.

Because static method is bound to class, on the other hand method is bound to object.

Can we override Java main

method?

(Ans: No, because main is a static method.)

Ques: What is the output?

; "method concat" = ans



CamScanner

Overriding Vs Overloading

- Don't Confuse the Concepts of Overloading and overriding (that is to say) the two
- Overloading deals with multiple methods with the same name in the same class, but with different signatures.
- Overloading deals with two methods, one in a parent class and one in a child class that have the same signature.
- Overloading lets you define a similar operation in different ways for different data.
- Overriding lets you define a similar operation in different ways for different object types.

Q.

Overloading example

```
class public class Clazz overload {  
    void add (int a, int b) {  
        System.out.println(a+b);  
    }  
    void add (int a, int b, int c) {  
        System.out.println (a+b+c);  
    }  
    void add () {  
        System.out.println ("Nothing to add");  
    }  
}
```

Class n't commits w/ self overriding
transf. with new transf. in mother
class
- except friend

Super keyword 3 fast mode

(super expression) immediate object creation

→ Super keyword जैसा होता है = ०१ - २३

- Aim: "super" keyword used to refer immediate super class object.

→ Use of super keyword in Java

1. Use of super with variables

2. Use of super with methods

3. Use of super with constructors

constructor glenot

Constructors

{ constructor mode

Exam
Variables

class Var { main bio

int number = 140; } १५०

Class 2. Car extends Var {

{ main dement mode १००
number = 120; }

void display() { main bio

System.out.println("Max Speed: "

+ super.number; }
Upgrate bio

CS

CamScanner

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

 for (String s1 = new String ("") : null
 new . display (s1))

 long n : longest output to get the

 additional other string number = 160

abstraction

Example Methods

overloading

Class Person {

 void mem () { } methods

Class S , O , P { " This is person class long "

 } } no abstractions used in program

Class Student extends Person {

 void mem () { } overrides base

 " Now we are " S , O , P " This is student class "

 void display ()

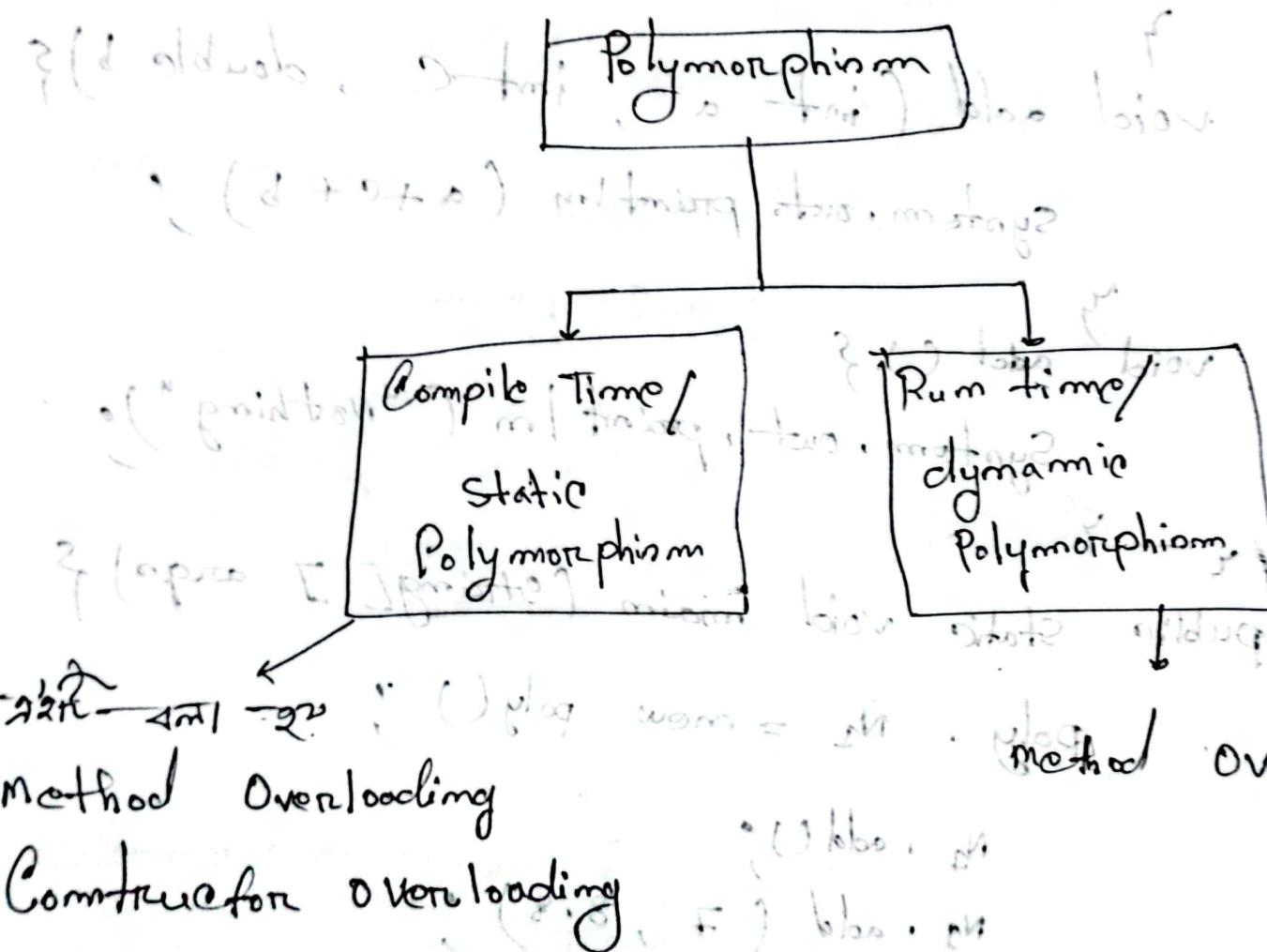
}



CamScanner

3. Polymorphism

→ Polymorphism means ability to have many different forms; "One interface, multiple methods". i.e. (d + o) returning diff. methods



~~and hence~~ → polymorphism is also known as PAM. i.e., it is a must

1. Compile-time polymorphism

2. Runtime polymorphism

3. Compile-time Polymorphism

Class → Cal

int Add (int a, int b) {

 return a+b; } q02

int Add (int a, int b, int c) {

 return a+b+c; } q02

double Add (double a, double b) {

 return (a+b)/2; } q02

public class Test {

 public void main (String args) {

 Cal S1 = new Cal();

 System.out.println ("sum " + cal.add(2, 3));

 System.out.println ("sum2 " + cal.add(1, 2, 4));

 System.out.println ("sum3 " + cal.add (2.2, 1.2));



CamScanner

1. WAP in Java to print odd numbers from 1 to n

```
import java.util.Scanner;
public class OddNumf
public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.print("Enter n : ");
    int n = scanner.nextInt();
    System.out.println("odd num " + n);
    for (int i = 1; i <= n; i++) {
        if (i % 2 != 0)
            System.out.println(i);
    }
}
```

2. WAP. in Java to find the factorial
of a given number

import java.util.Scanner;

public class Factorial

{
 String s = args[0];

 Scanner sc = new Scanner(s);

 int num = sc.nextInt();

 int fact = 1;

 for (int i = 1; i <= num; i++)

 fact *= i;

 System.out.println("Factorial of " + num + " is " + fact);

 System.out.println("Factorial of " + num + " is " + fact);

CamScanner

4. Abstraction

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

⇒ How to achieve Abstraction?

There are two ways to achieve abstraction in Java.

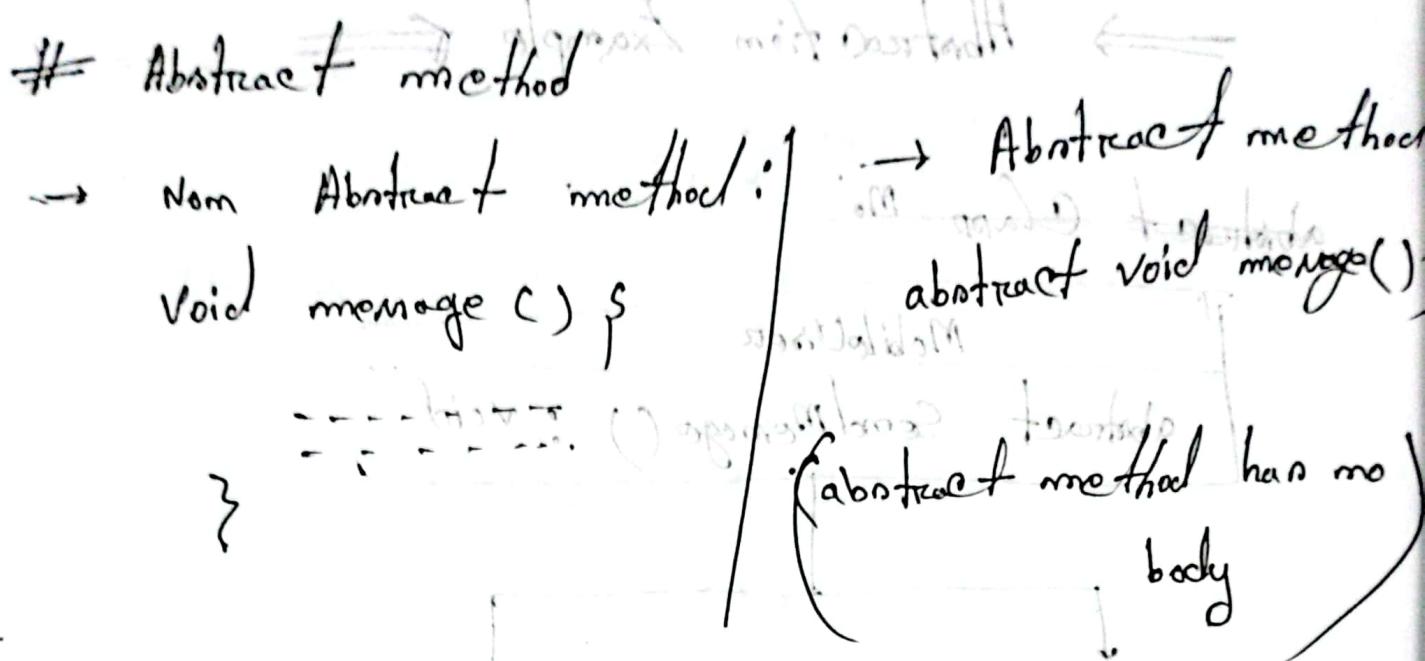
1. Abstract class (0 to 100% abstraction)
2. Interface (Achieve 100% abstraction)

Abstract Class

→ easiest way to implement inheritance
class mob { fast and strong }

→ Abstract Class from -> Object Class

→ easier Abstract Class →
abstract class mob { }



→ Abstract class have abstract and non abstract methods

Example

```

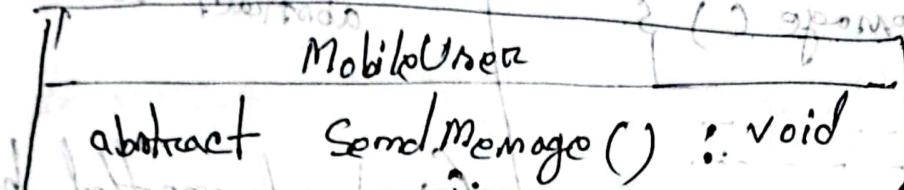
abstract class Mobile {
    void call() {
        System.out.println("Hello");
    }
}

abstract void sendMsg();
  
```

all methods in abstract class must implement two methods

====> Abstraction Example

~~abstract class~~ MobileUser



abstract class MobileUser

abstract void sendMessage()

class Rahim extends MobileUser

@Override void sendMessage()

void sendMessage()

System.out.println("Hi, this is Rahim")

class Karim extends MobileUser

@Override

void sendMessage()

SOP ("Hi, this is Karim")



CamScanner

public class Test {
 public static void main(String[] args) {

mobileUser mp;

mp = new Rahim();

mp.sendMessage();

mp = new Karim();

mp.sendMessage();

and sets from bottom, name

becomes) bottom of message top of ; sibling

, and bottom; bottom child

prints and function message top of ; sibling

, bottom; name

and from bottom

, value goes whatever form of name ; below

→ you are tasked with developing a payment processing system for an e-commerce platform. The system must handle different types of payments, including Credit Card Payments, Bank Transfers, and Digital Wallet Payments. To simplify the process and maintain a clean interface for payment processing, use method overloading to implement multiple versions of a processPayment method in a class called PaymentProcessor, why would method overloading be more effective in this scenario compared to using different method names?

N.B. Process credit card payments using a credit card, requiring a credit card number, expiration date, and CVV. Process bank transfers using bank account details (account number and routing number) and process digital wallet payments using the wallet's email address and payment amount.

Code

class PaymentProcessor

{
 void processPayment(String creditCardNumber,
 String expirationDate,
 String cvv)
 {
 // logic here
 }
}

System.out.println("Processing credit card")

method algorithm framework of Payment -> Visitor

Visitor pattern in bottom ConcreteVisitors of

public void processPayment (String accountNumber,
 String routingNumber) {
 // logic of banking
}

System.out.println("Processing bank transfer...")

processBankTransfer is final class of ConcreteVisitors of Visitor

public void processPayment (String emailId, double amount)

objects transfer and print statement and AbstractFactory

System.out.println("Processing digital
wallet payment - - -")

using Factory with factory automating below logic

functions - transferring has methods

public class Main {

 public static void main (String [] args) {

 PaymentProcessor processor = new PaymentProcessor();

 processor.processPayment ("12345678-9876-5432",

 "12/23", "123");

 processor.processPayment ("1234-47859910", "98765");

 processor.processPayment ("user@gmail.com", 106500.75);

y

y

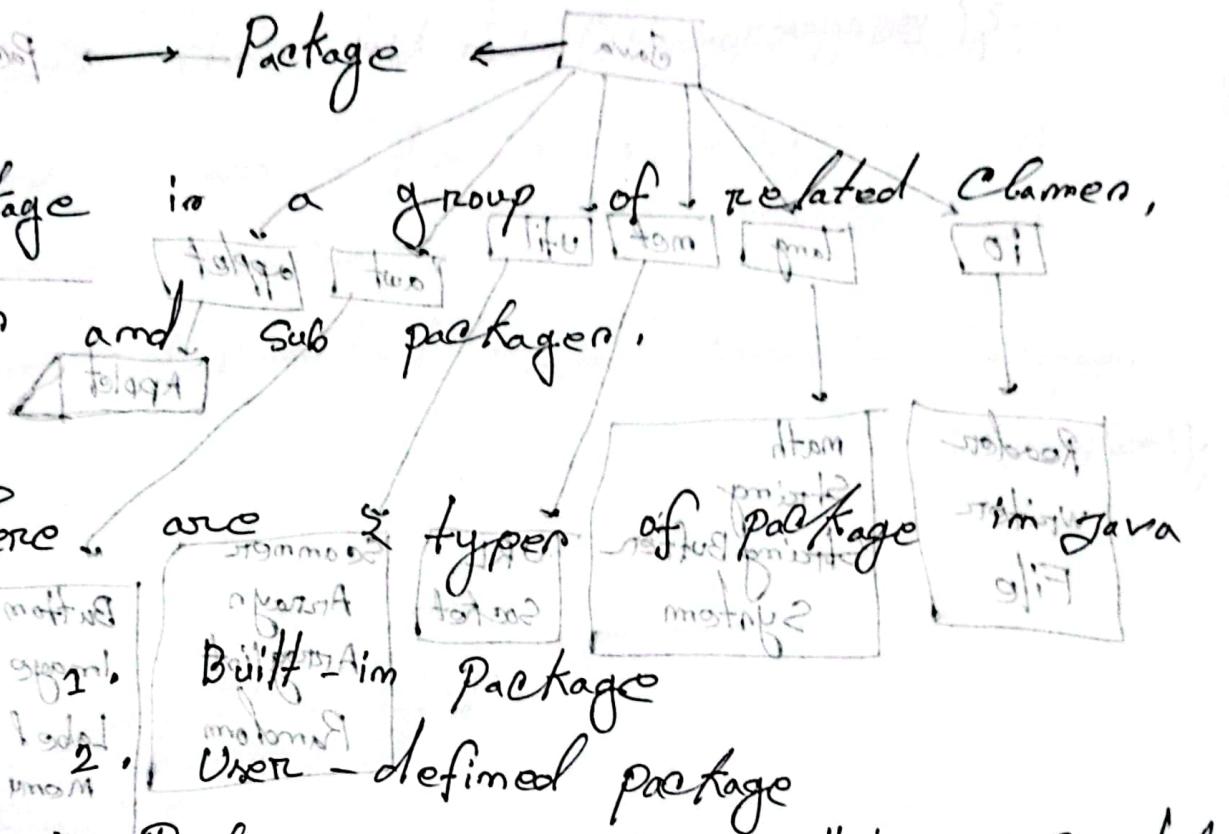
Class Note Final

→ Object Oriented Programming Final Slides ←

slide - 8

Object → Package

- A package is a group of related classes, interfaces and sub packages.



→ There are 2 types of package in Java



1. User-defined package

1. Built-in Packages are packages that are created that we will only use

Example import java.util.Scanner;

2. User-defined package is the things we make ourselves

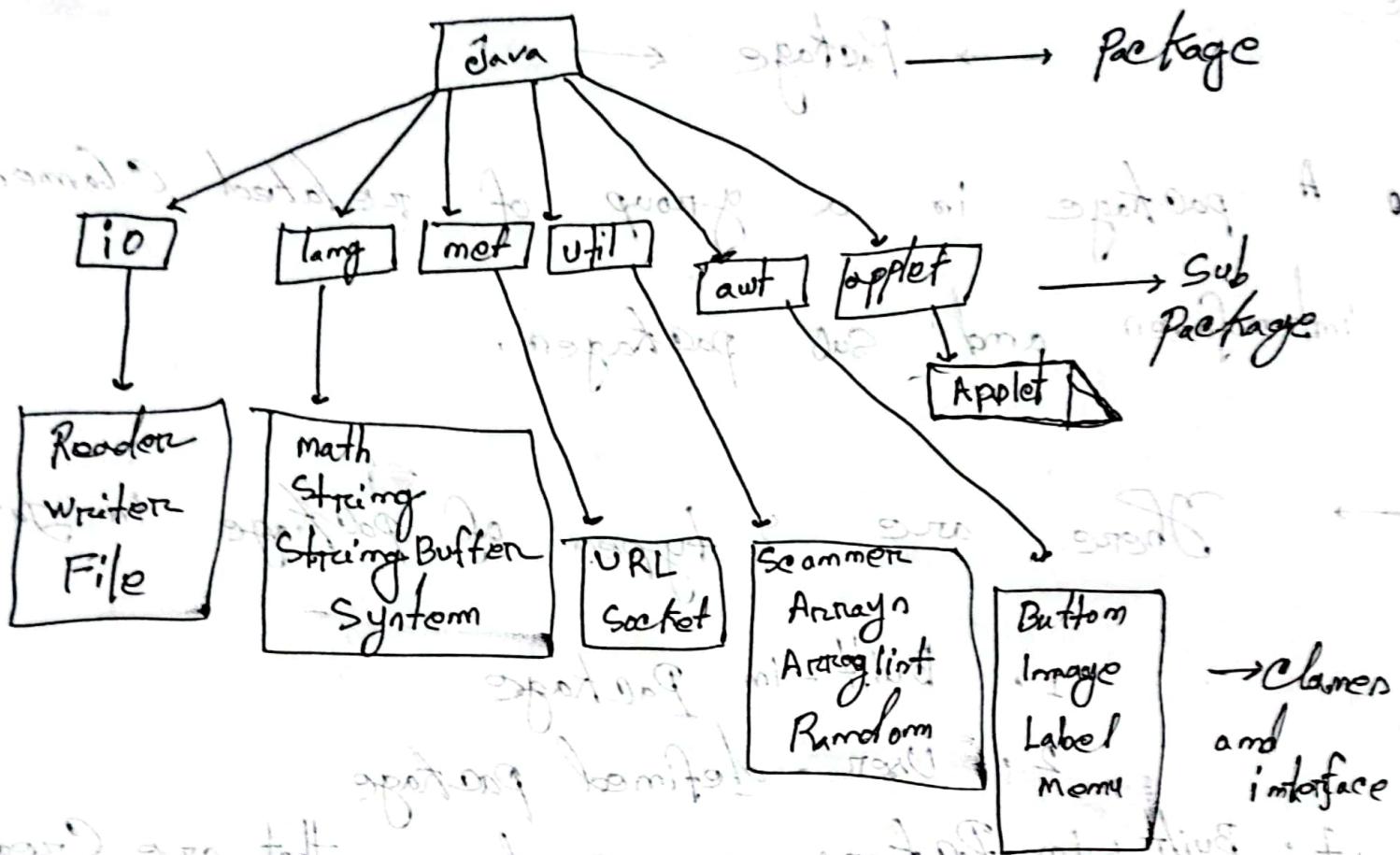
(Math, Util, Tom)

both anything with prefix are aspects in this

• All of the parts come

• As in new Util, new Tom, new Matrix

→ Built - in - package



Example

```
import java.util.Scanner;
```

```
import java.lang.Math;
```

```
import java.io.Reader;
```

```
import java.util.*;
```

→ Built - in packages are existing java packages that come along with the JDK.

* example : `java.lang`, `java.util`, `java.io` etc.

```
import java.util.Arrays;
public class MyClass {
    public static void main(String args[]) {
        int[] arr = {10, 12, 8, 9};
        Arrays.sort(arr);
        System.out.println("Sorted Array is " + Arrays.toString(arr));
    }
}
```

ex

```
import java.util.ArrayList;
class ArrayList {
    public static void main(String args[]) {
        ArrayList<Integer> myList = new ArrayList<Integer>();
        myList.add(3);
        myList.add(2);
        myList.add(1);
    }
}
```

→ User-defined package

This also allows you to create packages as per your need. These packages are called user-defined packages.

To define a package in Java, you use keyword `package`.

```
package packageName;
```

```
package Com.tenk;
```

```
{public static void main (String [] args) {  
    System.out.println ("Hello");  
}}
```

y
y

→ How to import packages in Java? 3-9670
Tracing

Java has import statement that allows you to import an entire package (as in earlier examples), or use only certain classes and interfaces defined in the package.

For example,

```
import java.util.Date;
```

↳ imports all classes in java.util package. A class can also be imported.

↳ imports all symbols of Date class in java.util package.

↳ imports all symbols from Date class in java.util package.

↳ imports all symbols from Date class in java.util package.

↳ imports all symbols from Date class in java.util package.



CamScanner

slide-2 ~~Contract~~ mi report do pogrami of with a

→ Java interface ~~is~~ ~~not~~ ~~has~~ ~~methods~~ ~~program~~ ~~and~~ ~~can't~~

→ implements keyword in Interface.

→ Why use Interface? → methods who are the

→ extends keyword in Interface.

→ Abstract Class vs Interface.

→ → Java Interface ← ←

A Java Interface is a programming contract in Java that acts as a blueprint or contract. It is used to define a set of methods that a class must implement ensuring a common structure across multiple classes.

→ Example of Java Interface in Java

interface polygon {

 public void getArea();

algorithm to calculate area of semi

→ Hence, polygon is an interface even though it does not have any method. We use the interface keyword to declare an interface.

→ abstract method getArea() must be declared in interface.

Definition of Java Interface

→ An interface is a reference type in Java.

Java interface : Unidirectional class

→ It can only contain abstract methods (methods without a body) and static final fields (constants).

→ Interfaces are used to achieve multiple inheritance and abstraction in Java.

why Use an Interface

& mapping constraint

1. Multiple Inheritance via coding

Since Java does not support multiple inheritance through classes or interfaces, provides a way to achieve it.

2. Abstraction

Overriding methods & Interface enforce a structure by

ensuring that implementing classes must implement defined certain methods.

3. Code Reusability

Interfaces allow you to define shared behaviors among multiple classes.

(automatically) abstract

avoids code repetition

using inheritance

→ Example of an Interface in Java
interface Animal { void eat(); void sleep(); }

void eat(); void sleep();

class Dog implements Animal { @override

public void eat() { System.out.println("Dog is eating"); }

@override

public void sleep() {

System.out.println("Dog is sleeping"); }

public class Test { public static void main(String[] args) {

Animal s = new Dog();

dog.eat(); // Output: Dog is eating

dog.sleep(); // Output: Dog is sleeping

(Dog.class + " is running now") will print two messages

→ implements keyword in Interface ←

- like abstract classes, we can't create objects of interface.
- however, we can implement interfaces in other classes.
- In Java, we use the implements keyword to implement interfaces.

example

interface Student {

 void get();

class Teacher implements Student {

 public void getStudent() {

 int ID = 2088; String name = "Sojib";

 System.out.println("Student" + ID, name);

 }

}

CS CamScanner

```
class Teacher {
    void teach() {
        System.out.println("Teaching");
    }
}

public static void main (String [] args) {
    Teacher teacher = new Teacher();
    teacher.teach();
    System.out.println(teacher.getStudent());
}
```

→ Extends keyword in interface

→ Similar to classes, interfaces can extend other interfaces

→ The extends keyword is used for extending one interface, whereas limit is required for multiple

example interface Animal {

```
    void eat();
    void sleep();
}

interface Dog extends Animal {
    void bark();
}
```

→ Dog class will implement both methods

→ Abstract Class ✓ Interface ←
↳ (Aggregates) more like object oriented
Interface → more like Abstract Class

- + Can only have abstract methods.
- Can have abstract and non abstract methods.

- 2. It supports multiple inheritance.
- It doesn't support multiple inheritance.

- 3. Can only have static and final variables.
- Can have static, non static, final and non final variables.

- 4. Fully abstract
- Partial, abstract form

5. Example of Animal abstract interface Animals

```
void eat();
```

Y

5. Example of abstract class Animals

```
abstract void eat();
```

Y

→ Exception ←

- Java Exception

- Types of Exception

- Exception & Handling

- Example : Java Exception Handling

Java Exception

In Java, an exception is an unexpected event or error that occurs during the execution of a program, disrupting its normal flow. Exceptions are objects that represent these errors, and Java provides a robust mechanism to handle them, ensuring that the program does not crash abruptly and can recover gracefully.

Type of Exception

1. Runtime Exception

2. IO Exception

1. → Runtime Exception

A runtime exception happens due to a programming

error. They are also known as unchecked

exception

These exceptions are not checked at

Compile-time but run-time. Some of them

Common runtime exceptions are

• Improper use of an API
• null pointer access, math errors

• out-of-bounds array access

• Dividing a number by 0

2. IO Exception

An IO Exception is also known as a checked exception. They are checked by the compiler at the compile-time and the programmer is prompted to handle these exceptions.

Some of the examples of checked exception are:

- Trying to open a file that doesn't exist results in FileNotFoundException.
- Trying to read past the end of a file.

Exceptions are useful utilities of Java to handle errors.

Exception is an error that occurs during the execution of a program. It is used to handle errors.

Java Exception Handling

Java provides the following keywords for exception handling:

1. try

→ Defines a block of code to monitor

for exception to handle it.

2. catch

→ Defines a block of code to handle the exception.

3. finally

→ Defines a block of code that executes

whether an exception occurs or not.

4. throw

→ Used to explicitly throw an exception.

5. throws

→ Used in method declarations to specify exceptions that a method might throw.

→ Example of Exception Handling

```
import java.util.*;  
class Main {  
    public static void main(String[] args) {  
        try {  
            System.out.println("Inside try block");  
            int a = 10 / 0;  
            System.out.println("Rest of code in try block");  
        } catch (ArithmaticException e) {  
            System.out.println("A => " + e.getMessage());  
        } finally {  
            System.out.println("Finally block");  
        }  
    }  
}
```

→ Java try...Catch Example

```
class Main {  
    public static void main(String[] args) {  
        try {  
            int divideByZero = 10 / 0;  
            System.out.println("Rest of code in try block");  
        } catch (ArithmaticException e) {  
            System.out.println("A => " + e.getMessage());  
        }  
    }  
}
```

Example: ~~praktisch~~ ~~mitgebracht~~ ~~zu überprüfen~~
Main Main {
 public static void main (String args) {
 try {
 int di = Integer.parseInt (args[0]);
 System.out.println ("Finally block ");
 } catch (Exception e) {
 System.out.println ("Error ");
 } finally {
 System.out.println ("Finally block ");
 }
 }
}

→ Java try...Catch...finally Block

```
import java.io.*;
class L {
    private int[] list = {5, 6, 7, 10, 2}; // missing null
    public void writeList() {
        PrintWriter out = null;
        try {
            System.out.println("Entering try");
            out = new PrintWriter(new FileWriter("outputFile.txt"));
            for (int i=0; i<7; i++) {
                out.println("Value at " + i + " = " + list[i]);
            }
        } catch (Exception e) {
            System.out.println("Exception => " + e.getMessage());
        } finally {
            if (out != null) {
                System.out.println("Closing");
                out.close();
            }
        }
    }
}
```

else {
 void flatten(NoData* node);
 sop ("pre mo");

class main {
 {f.s.o, F.i.d, o} = tail [This string

p s v m (string[7] org) }
; there are two methods

L. ("S1 = new L();

S1, writeln(L); returning two methods

("Method1") writeln(L) returning one method = true

} (t+i) (f-i) (o=i tail) and

• (L.tail + " + i + " ; true v *) returning false

} (o methods) total

(("Method1", o + " (= methods)) returning two methods

} (true = ! true) fi

→ Multiple Catch blocks

→ priority ←

class L {

public int[] arr = new int[10];

for (int i = 0; i < arr.length; i++)

public void write() {

try {

arr[10] = 11;

catch (Exception e1) {

System.out.println("No. " + e1.getMessage());

catch (IndexOutOfBoundsException e2) {

System.out.println("Index " + e2.getMessage());

class Main {

public static void main(String[] args) {

L l = new L();

l.write();

}



CamScanner

→ String ←

→ standard data type → standard data structure → standard data algorithm →

→ String is a class in Java, which is immutable and holds a sequence of characters. It comes under the Java.lang package.

= Declaring Strings in Java

1. Using String Literals:

Example

String str1 = "Hi";

String str2 = "Hello".

2. Using the new keyword:

String str = new String("Hello");

↳ Question : ?

→ String Methods ←

String charAt()

Example : public class CharAtExa {

psvm (String args[]) {

String name = "Soyib",

Char ch = name.charAt(3);

sop(ch);

Applet is a type of program that is executed in the browser to generate the dynamic content. It

runs inside the browser and works at client side.

String length()

Example :

public class L {

psvm () {

String N₁ = "Soyib",

String N₂ = "Shahjalal",

sop ("S length " + N₁.length());

sop ("S " + " " + N₂.length());

→ Content ← ~~content~~ Applet programming

- Java Applet
- Advantage of Applet
- Hierarchy of Applet
- Lifecycle of Java Applet
- Simple example of Applet

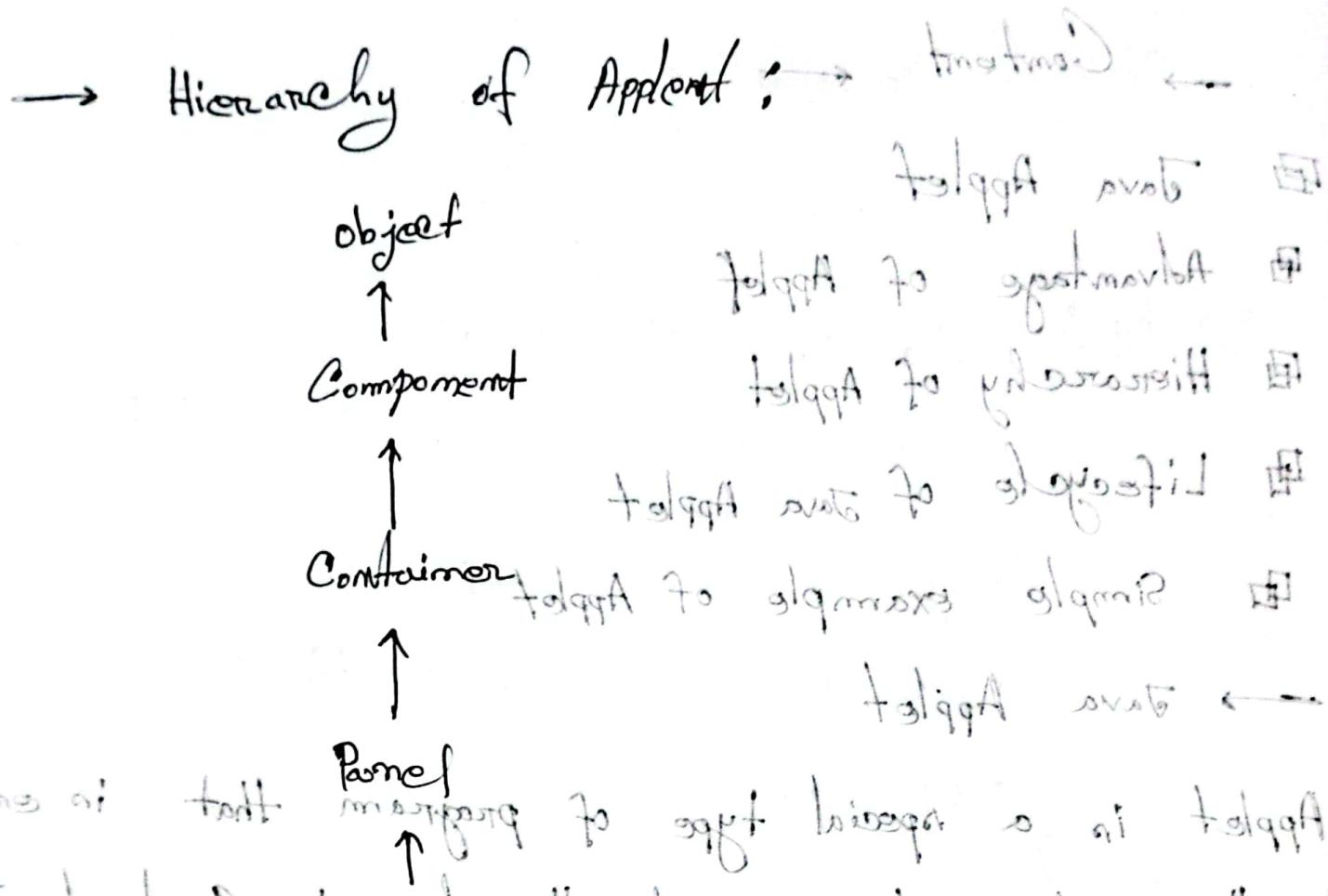
→ Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

→ Advantage of Applet

There are many advantages of applet, they are as follows:

- # It works at client side so less response time.
- # Secured
- # It can be executed by browser running under many platforms, including Linux, windows, mac os etc.



→ Lifecycle of Java Applet

- Applet is initialized
- u Started to others
- u Painted
- u Stopped

• u Destroyed and no HI

• u Redraws and repaints

→ Simple example of Applet

```
import java.applet.Applet;
import java.awt.Graphics;
```

public class First extends Applet {

```
public void paint(Graphics g) {
    g.drawString("welcome", 150, 150);
```

Yours to bring program in printed off

metting

interest, printed not electronic version and write

• edit printed from bridge

• changes

(and print off too) ; off, i, print bring

• ("for example") off won't bring off

(example off page)

Content ~~of~~ To ~~of~~ ~~more~~ ~~skip~~ ←

- Java File Handling ~~of~~ ~~page~~, ~~not~~ ~~begin~~
 - Methods of File, ~~char~~, ~~two~~, ~~no~~ ~~begin~~
 - Java Create, and write To File
 - Get File Information
 - Java Delete File
- (~~class~~, ~~new~~) ~~using~~ ~~bio~~ ~~file~~
→ Java File Handling ←

File handling is an important part of any application.

Java has several methods for Creating, reading, updating, and deleting files.

Example:

```
import java.io.File; (Import the File Class)
```

```
File myobj = new File ("filename.txt");
```

(Specify the filename)

→ The File Class has many useful methods for creating and getting information about file.

| <u>Method</u> | <u>Type</u> | <u>Description</u> |
|-------------------|-------------|--------------------------------------------|
| canRead() | Boolean | Tests whether the file is readable or not. |
| canWrite() | Boolean | Tests whether the file is writable or not. |
| CreateNewFile() | Boolean | Creates an empty file. |
| getName() | String | Returns the name of the file. |
| getAbsolutePath() | | |
| length() | Long | |
| list() | String[] | |
| mkdir() | Boolean | |
| delete() | Boolean | |
| exists() | Boolean | |
| isDirectory() | Boolean | |
| isFile() | Boolean | |

→ Java Create and write To File ←

To Create a file in Java, you can use the `createNewFile()` method. This method returns a boolean value, true if the file was successfully created, false if the file already exists. Note that the method is enclosed in a try...catch block. This is necessary because it throws an `IOException` if an error occurs.

→ Create File -

```
import java.io.File;
```

```
import java.io.IOException;
```

```
public class CreateFile
```

```
    public static void main (String [] args)
```

```
        try {
```

```
            File s1 = new File ("filename.txt");
```

```
            if (s1.createNewFile ()) {
```

```
                System.out.println ("File "+s1.getName());
```

```
            } else {
```

```
                System.out.println ("File al ex");
```

```
        } catch (IOException e) {
```

```
            System.out.println ("An error oc");
```



→ Write To a File

In the following example, we use the FileWriter class

together with its write() method to write some text to the file we created, in the example above. Note that when you are done writing to the file, you should close it with the close() method.

```
import java.io.FileWriter;
import java.io.IOException;
public class WriteToFile {
    public static void main(String[] args) {
        try {
            FileWriter f = new FileWriter("filename.txt");
            f.write("File in Java");
            f.close();
            System.out.println("Successfully");
        } catch (IOException e) {
            System.out.println("An error");
            e.printStackTrace();
        }
    }
}
```

→ Java Read File ← It's time

In the previous chapter, you learned how to

Create and write to a file. Now after this reading

In the following example, we are using the Scanner
class to read the contents of the text file which
created in the previous chapter.

Example

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReadFile {
    public static void main(String[] args) {
        try {
            File S1 = new File ("filename.txt");
            Scanner S2 = new Scanner (S1);
            while (S2.hasNextLine ()) {
                String data = S2.nextLine ();
                System.out.println (data);
            }
            S2.close ();
        }
    }
}
```

catch (FileNotFoundException ex) {
 System.out.println("File not found error occurred");
}

else {
 System.out.println("File found");
}

e. printStackTree():

• prints the stack tree
• stack tree is a binary tree where
 left child is always null

• right child is either
 null or a stack tree

• left command(P^N) will return a new stack tree
 with root as P

• command($\text{A} + \text{B}$) will return a stack tree
 with root as + and A and B as children

• command($\text{A} * \text{B}$) will return a stack tree
 with root as * and A and B as children

• command(A / B) will return a stack tree
 with root as / and A and B as children

→ Get File Information (Information about file)

To get more information about a file use any of the File methods.

Ex

```
import java.io.File;
```

```
public class L {  
    public ( ) {
```

```
        File N, = new File ("filename.txt");
```

```
        if ( s, . exists ()) {
```

```
            System.out.println ("File name" + s, . getName ());
```

```
            System.out.println ("A" + s, . getAbsolutePath ());
```

```
        } else {
```

```
            System.out.println ("The file does not ");
```

```
}
```

```
}
```

```
b
```

→ Java Delete File

To delete a file in java, use the delete() method.

example

```
import java.io.File;
```

```
public class Deletefile {
```

```
    public static void main ( ) {
```

```
        File s1 = new File ("filename.txt");
```

```
        if (s1.delete ()) {
```

```
            System.out.println ("File deleted");
```

```
        } else {
```

```
            System.out.println ("Failed to delete ");
```

```
        }
```

```
    }
```

```
}
```

→ Delete a Folder

function () starts with

you can also delete many folder. However it must be empty:

Example

```
import java.io.File;
public class Delete {
    public static void main(String[] args) {
        File s1 = new File("C:\Users\myName\Desktop");
        if (s1.delete()) {
            System.out.println("Deleted " + s1.getName());
        } else {
            System.out.println("File not found");
        }
    }
}
```

- ~~Thread~~ → Thread →
→ Multithreaded Programming →
→ When a Java program starts up, one thread begins running immediately.
→ Other child threads are spawned from the main thread.
→ often it must be the last thread to finish execution.

Useful Methods of Thread.

1. static Thread CurrentThread()
→ returns a reference to the thread in which it is called.
2. static void sleep (long milliseconds)
→ cause the thread from which it is called to suspend execution for the specified period of milliseconds.

3. final void setName(String threadName)

→ set the name of the thread from which it is called to the name given as parameter.

4. final String getName()

→ return the name of the thread from which it is called.

Creating Threads

→ Two ways

1. Implement the Runnable Interface:

2. Extend the Thread Class.

↳ Definition (and) code how threads work at the lowest level. It's difficult to understand so many things like inheritance of multiple classes of a class.

Q) First Approach - Implement Runnable

→ Create a class that will implement Runnable.

To implement Runnable, the class need only to implement:

public void run()

→ Establishes the entry point for another, concurrent thread.

→ Introduce an object of type Thread from within the same class.

→ Call the Start() method of the new object of type Thread.

(is implemented by the class)

("background thread") entering two methods

(*, because local primitive *) goes @

Example - First Approach

```
class NewThread implements Runnable {  
    public void run() {  
        Thread t = new Thread(this, "demo thread");  
        System.out.println("child thread: " + t);  
  
        try {  
            for (int m = 0; m > 0; m--) {  
                System.out.println("child thread: " + m);  
                Thread.sleep(100);  
            }  
        } catch (InterruptedException e) {  
            System.out.println("child interrupted.");  
        }  
        System.out.println("Exiting child thread.");  
    }  
}
```

```
class ThreadDemo {  
    public static void main (String args) {
```

```
        new Thread (new Runnable () {  
            public void run () {
```

```
                try {
```

```
                    for (int m = 5; m > 0; m--) {
```

```
                        System.out.println ("main thread : " + m);
```

```
                        Thread.sleep (1000);
```

```
                } catch (InterruptedException e) {
```

```
                    e.printStackTrace ();
```

```
                    System.out.println ("main ");
```

```
        } catch (Exception e) {
```

```
            e.printStackTrace ();
```

```
        }
```

```
    } catch (Exception e) {
```

```
        e.printStackTrace ();
```



CamScanner

■ Second Approach - Extends Thread

→ Create a new class that extends Thread

and then create an instance of that class.

→ The extending class must override the run() method.

→ It must also call the start() method to begin execution of the new thread.

Example: NewThread extends Thread

class NewThread extends Thread {

 NewThread() { print("In constructor of NewThread") }

 String super(Thread thread) { print("In constructor of NewThread") }

 void run() { System.out.println("Child Thread: " + this); }

 void start() { print("In start of NewThread") }

 void join() { print("In join of NewThread") }

 void interrupt() { print("In interrupt of NewThread") }

print("In constructor of NewThread")

print("In start of NewThread")

print("In join of NewThread")

print("In interrupt of NewThread")

print("In destroy of NewThread")

print("In destroy of NewThread")

print("In destroy of NewThread")

print("In destroy of NewThread")

Creating Multiple Thread

In Java multiple threads (means running multiple tasks concurrently using separate threads). A thread is a lightweight subprogram that enables a program to perform multitasking, where different tasks can run independently and simultaneously.

What is Multiple Threading

- Multiple threading involves running two or more threads simultaneously within a single program.
- It allows efficient utilization of the CPU by performing tasks concurrently.
- Threads Share the same memory space, making inter-thread communication easier and faster.

- why use multiple threads?
- Improved Performance; Enables parallel processing, making programs faster.
 - Concurrency; Executing multiple tasks simultaneously, improving responsiveness.
 - Shared Resources
 - Responsiveness

Multiple Example

```
class Multiple implements Runnable {  
    String name;  
    Thread t;  
    public Multiple (String tmp) {  
        name = tmp;  
    }  
    public void run() {  
        for (int i = 0; i < 5; i++) {  
            System.out.println ("New thr." + i);  
        }  
    }  
}
```

```
public void run() {
    algorithm sum();
    try {
        for (int i = 0; i < 1000; i++) {
            System.out.println(name + " : " + i);
            Thread.sleep(1000);
        }
    } catch (InterruptedException e) {
        System.out.println("Exception handled");
    }
}
```

```
class Java {
    public sum() {
        algorithm sum();
        new Multiple("One");
        new Multiple("Two");
        new Multiple("Three");
    }
}
```

(return sum())
return sum();

(++ "I am a")



CamScanner

→ `inAlive()`:

- final boolean `inAlive()` - used for flags monitoring
- it returns true if the thread upon which it is called is still running.

→ `join()`

- final void `join()` throws InterruptedException
- This method waits until the thread on which it is called terminates.



CamScanner

Thread Priority

- Higher - priority threads get more CPU time than lower - priority threads (Condition applies when there is no thread of H priority)
- final int getPriority ()
- final int setPriority (int level) (It must be level between MIN_PRIORITY (1) and MAX_PRIORITY (5); if not)
- Default priority is NORM_PRIORITY which is 5.

■ Thread Synchronization ■

- Two or more threads need access to shared resource.
- Need some way to ensure that the resource will be used by only one thread at a time.
- Synchronization
- Monitor
- A monitor is object that is used as a mutually exclusive lock.
- Only one thread can own a monitor at a given time.
- Threads attempting to enter a lock monitor will be suspended until first thread exits.
- Other threads are waiting for the monitor.

Inter Thread Communication

- wait(): tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify().
- notify(): wakes up a thread that called wait() on the same object.
- notifyAll(): wakes up all the threads that called wait() on the same object. One of the threads will omit moving to minimum granted accm.