Core Java notes

What is Class ,Methods and objects?

**Class**

A class is a blue print or template in java which consists of collections of methods and objects.

package org.emp;

public **class** **Employee** {

public void empId() {

System.*out*.println("enter emp id");

}

**Note**: Starting word should be in upper case

**Methods**

An action which is performed inside the class

**Example prog:**

public class Employee {

**public void empId()** {

System.***out***.println("enter emp id");

}

**public** **void** **empName()** {

System.***out***.println("enter employee name");

}

**Note**: Starting word should be in starts with lower case

Every next word should starts with Uppercase.

**Objects**

We can call methods from class using objects

It’s an item created from the class to call methods. It stores heap memory available in the JVM

**public static void main(String[] args) {**

**System.*out*.println(" employee details");**

**Employee details = new Employee ();**

**details.empId();**

**details.empEmail();**

**}**

Here the object is **details.**

**Note:** Starts with lower case

**DataTypes:**

Data type tells Java what kind of data a variable will store.

DataType Size WrapperClass

byte 1 Byte

short 2 Short

int 4 Integer - use whole numbers

long 8 Long – no decimals are used and in ending “l” should be used

float 4 Float - f -> uses decimals end f should be sued

double 8 Double -> d-> infinite

WrapperClass

A wrapper class is a class version of a primitive data type.

public class Main {

public static void main(String[] args) {

// primitive

**int num = 10;**

**// wrapper class**

**Integer numObj = num;** // auto-boxing

System.out.println(numObj); // Output: 10

}

}

Inheritance:

Can access one class methods from another class by using keyword word called extends

There are 4 types of inheritance:

1) Single Inheritance

2) Multiple Inheritance-This cannot be achieved from java but we can achieve it in Interface

3) Multilevel Inheritance

4) Hierarchical Inheritance

**Single inheritance**

A combination of one parent class and child class

**Example prog**

**// Class A -parent**

Public class Classroom {

public void study() {

System.out.println("All students study in the classroom");

}

}

**// Class B -child**

Public class Student extends Classroom {

Public void play() {

System.out.println("The student also plays during break time");

}

}

public class Main {

public static void main(String[] args) {

Student s1 = new Student();

s1.study(); // Inherited from ClassA

s1.play(); // CHILD METhod

}

}

**Multilevel Inheritance:**

It is a tree like structure.

A class is being inherited from a class from more than 1 parent

// Grandparent class

class Grandfather {

void hasBicycle() {

System.out.println("Grandfather has a bicycle");

}

}

// Parent class

class Father extends Grandfather {

void hasCar() {

System.out.println("Father has a car");

}

}

// Child class

class Son extends Father {

void hasBike() {

System.out.println("Son has a bike");

}

}

// Main class

public class Main {

public static void main(String[] args) {

Son s = new Son(); // Create object of child class

s.hasBicycle(); // From Grandfather

s.hasCar(); // From Father

s.hasBike(); // From Son

}

}

**Hierarchical Inheritance**

**One parent is being used for more number of childrens**

**Animal**

**package** org.test;

**class** Animal {

**public** **void** breathe() {

System.***out***.println("All animals breathe");

}

}

**Dog**

**package** org.test;

public **class** Dog **extends** Animal {

**public** **void** bark() {

System.***out***.println("Dog barks");

}

}

**Cat**

**package** org.test;

**public** **class** Cat **extends** Animal {

**public** **void** meow() {

System.***out***.println("Cat meows");

}

**public** **static** **void** main(String[] args) {

Dog d = **new** Dog();

d.breathe(); // From Animal

d.bark(); // From Dog

Cat c = **new** Cat();

c.breathe(); // From Animal

c.meow(); // From Cat

}

}

**output**

All animals breathe

Dog barks

All animals breathe

Cat meows

**Polymorphism**

**Executing methods more than once is known as polymorphism**

**There are 2 types**

**1)method overriding- different class but should have the same method type**

**2)method overloading -Single method but different arguments**

**Method overriding example program**

// Parent class

class Animal {

void sound() {

System.out.println("Animal makes a sound");

}

}

// Child class

class Dog extends Animal {

// Method overriding

@Override

void sound() {

System.out.println("Dog barks");

}

}

// Main class

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.sound(); // This will call the Dog's version

}

}

**Output**

Dog barks

This omits the parent class and delivers the child class

**methodOverloading**

class Greeter {

// Greet without name

void greet() {

System.out.println("Hello!");

}

// Greet with name

void greet(String name) {

System.out.println("Hello, " + name + "!");

}

// Greet with name and age

void greet(String name, int age) {

System.out.println("Hello, " + name + "! You are " + age + " years old.");

}

}

public class Main {

public static void main(String[] args) {

Greeter g = new Greeter();

g.greet(); // No argument

g.greet("Gopika"); // One argument

g.greet("Gopika", 25); // Two arguments

}

}

**output**

Hello!

Hello, Gopika!

Hello, Gopika! You are 25 years old.

**Scanner class**

Get the input from JAVA user at the run time

The Scanner class has present package called

java.util-> predefined package

import java.util.Scanner;

public class ScannerClass {

public static void main(String[] args) {

System.out.println("Start Entering the Values");

Scanner sc = new Scanner(System.in);

int empId = sc.nextInt();

System.out.println("The Value is " +empId);

}}

**Abstraction**

There are 2 types of abstraction

1) Partial Abstarction- Abstarct class

2) Fully Abstarction -Interface

**Partial Abstraction**

Partial Abstraction means a class hides some details (abstract methods) but also includes somenormal methods.

Fully Abstraction