Day 1:

JAVA – Is a OOP language, platform Independent and also scalable.

Everything in Java is an Object.

Again we have classes where we write all our business logic etc..But objects are needed to work upon the classes.

To build web applications, Mobile applications etc..

Nowadays we have many built in frameworks which help us build the applications in a simple and robust way.

It is more structured so that it is maintenance easy.

Installation:

Download Java – JDK

Set environment variable

What are the OOP (Object Oriented Programming) concepts?

1. Abstraction
2. Encapsulation
3. Inheritance

Abstraction: Where I will not define things until required and only the necessary classes will do the implementation as per the logic needed. Hiding the implementation/details and not revealed to the outside world.

Abstract class policy{

Float calculatePolicyPremium();

}

Class JeevanBheem extends Policy{

Float calculatePolicyPremium(){

}

}

Class JeevanAnand extends Policy{

Float calculatePolicyPremium(){

}

}

Polymorphism:

One name many forms.

There are 2 types in polymorphism:

1. Method Overloading
2. Method Overriding
3. Method Overloading:
4. To satisfy method overloading, the methods should vary in the number of parameters or the datatype of the parameters.
5. The return value is not counted to meet the method overloading.

Class Example{

Int add(int a, int b){

}

Float add(int a, int b, int c){

}

Float add(float a, float b){

}

}

1. Method Overriding:

This comes up with your inheritance.

Class Parent{

Void display(){

// I am parent

}

}

Class Child extends Parent{

Void display(){ // overriding

// I am child

}

}

Child c = new child();

c.display(); // he will call child

suppose we commented the method (display()) in the child class

c.display() // parent will be called

Inheritance: // Reusability of the code, whatever Parent has flows down to the child class.

Class Parent{

Void buy(){

}

}

Class Child extends Parent{

// whatever the Parent buys in the above method is available to the child as well.

}

Class Car{

Model name; // Model is another class

float price;

String color;

String make;

}

Class BMW extends Car{

}

Class Audi extends Car{

}

In Java, everything Is an Object.

Simple example of Java class:

Class Student{

String name;

Int age;

String address;

String college;

//Subject subject;

Public static void main(String args[]){

}

}

Basic datatypes, Variables, Modifiers, Classes, objects

Primitives datatypes:

int, float, char, byte, long, double, String, boolean

int x = 5;

Reference data types:

These are user defined types.

Student, Employee, Banking -> Object types.

The class should contain a unique information. We need readability.

Class College{

String name;

Int dept\_count;

Department depts; -> another class // user defined type

}

Linkedlist :

Java also works by references.

Literals/Constants:

Some hard coded value for the variables.

X =5; str = “Manju”

Char c = ‘a’

pi = 3.14

Escape characters:

\n = newline operator

\t = tab

\b = backspace

Many keywords being supplied by Java:

New, return, void, String, int, instanceof, extends etc…

Variables: it’s a name given to the datatype to hold some data. Or it’s a named storage and can get manipulated.

Int x = 5; // x is a variable

String \_name24 = “CapG”; // is a valid variable

Different type of variables:

Local variables: This is the one which is defined inside a method or a block.

Instance variables: These are defined at the class level but outside the methods

Class/Static variables: These variables are declared using static keyword and these aren’t referenced through objects but rather through class name.

Access Modifiers:

There are 4 access modifiers:

Default: It is accessible within the package.

Public: visible to all!

Private: It is visible to only that class where it is defined.

Protected: visible to the package and also to the subclasses.

The other modifiers we have are:

static

final

abstract

Day 2:

Basic Operators, Conditional statements and Loops:

What is an operator? Which acts upon the variables and manipulates it!

Arithmetic operators, Relational operators, Bitwise operators, logical operators etc..

Int z = x+y;

X = 5

X++

++X

x- -

s.o.p(x++) 5

s.o.p(x) 6

s.o.p(++x) 6

for(int i=0;i<5;i++)

x = 5; Assignment operator

x+=5 ------🡪 x = x+5

x-=5 -------🡪 x = x-5

The below ones are the relational operators:

x==5; equality check

x!=5

x<=5

Bitwise operators

1100

0010

<<

>>

&&, !!, !

X&&Y

!(x&&Y)

Conditional / ternary operator: ? and :

(X==5) ? y=2 : y=3;

**Instanceof** operator:

Employee e = new Employee()

e is an instance of type Employee

e instanceof Student

Conditional statement:

If(profit > 10%){

s.o.p(“I am happy”)

} else{

s.o.p(“sorry!”)

}

If

Nested if

If – else

Switch statement

Its not compulsory for an if to have an else…

If(x==1){

}

If(x==2){

}

If(x==3){

} else{

Unknown number!

}

Int x = 5;

Switch (x){

case 1:

s.o.p(x);

case 2:

s.o.p(x);

case 5:

s.o.p(x);

default:

s.o.p(“default”);

}

Looping:

For()

While()

Do while()

When we need to iterate through multiple records for a similar sort of business logic.

Arrays: []

It is collection of homogeneous data.

Public static void main(String args[])

Scanner – args[0], args[1]

Int n[] = {1, 2, 3}

N is an array of type integers.

n.length = 3

n[0], n[1], n[2]

int a[];

a = new int[10]; // memory allocation happened…

a[] = {1, 2, 3, 4, 5}

a[0] = 1;

a[1] = 2;

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

s.o.p(a[i])

}

i = a[2]

For(int i: a){

s.o.p(i);

}

While and do – while

Int x = 0;

Int a[] = {1,2,3}

While( x< a.length){

s.o.p(a[x])

x++;

}

The only difference between while and do-while is do-while will check the condition post execution of the while loop body.

do{

if(x == 4) {

return somevalue;

break;

}

x++;

} while(x<5);

Break and continue in the loops.

Inner:{

}

Arrays, loop statements, conditional statements, instanceof, scope, access modifiers etc..

Day 3:

Strings, Polymorphism, Inheritance, Abstraction, Interface

String:

int, float, double, boolean, char

String – it is a class and it is in java.lang package.

s.o.p(name + “ ” + age + “ ” + salary)

name is of type string

age is int

salary is float

name + “ ” => considered as a String and + is used for concatenation…

String x = name + “ ” + age + “ ” + salary

s.o.p (x)

You can create String variable in below two ways:

String s = “Manju”; // if this name is already existing in the memory then the JVM will just return the reference of that preexisting variable to this s. new one is not created. Suppose it doesn’t exist(“Manju” constant not der in memory/heap) then new instance/object s is created.

String s1 = “Manju” // this takes the same reference as like ‘s’

s = “abc”

s.o.p(s1)

String s = new String(“Manju”)

Public class StringExample {

Psvm (String args[]){

String s = “ABC”;

String s1 = “ABC”;

s = “XYZ”;

s.o.p(s1);

String s3 = new String(“EFG”); // always creates a new object..

}

}

Arrays – length arr.length

String – length()

Substring()

Uppercase()

Lowercase()

Substring()

Replace()

Concat()

“ABC” -> 2nd position of this string ‘x’ replace

AxC

Find the people whose name has a ‘Singh’ anywhere in der names

Contains()

indexOf()

I have a name which contains “Rao”

indexOf(“Rao”)

Roa, RAo etc all these to replaced as “Rao”

Input field

“ ABC”.trim() != “ABC”

S = GOVINDSHARMA

x= substring(s, 7,9)

x = sha

Strings are immutable in nature.

Polymorphism:

One name many forms.

1. Compilation way (function overloading)
2. Runtime way (function overriding - inheritance)

Inheritance:

Class A{

}

Class B{

}

Class C extends A, B{ // doesn’t work

}

Class B extends A{

}

Class C extends B{

}

Inheritance – reusability.

Class Parent{

Int i;

Parent(){

I =5;

}

Void display(){

}

}

Class child extends Parent{

Int n;

Child(){

super();

n=2;

}

Class A{

}

Class B extends A{

}

Class C extends A{

}

Void display(){

Super.display();

}

}

Assignment: real time scenario of Abstraction and do-while()

Abstraction:

Hiding the implementation.

What are the rules for Abstraction? Goes along with inheritance

1. We define abstract class using abstract keyword
2. Its said abstract because there are some methods which don’t have implementation. Its compulsory for the derived class to give the definition if not even that derived class also becomes abstract!
3. Its possible to have both abstract and non abstract things in a abstract class.
4. Cannot create an OBJECT of abstract class, but reference of abstract class can be created.
5. Do you have constructor in abstract class? Yes we have and we can have default as well as parameterized constructors.

abstract class A{

}

Class B extends A{

}

A a = new A(); // not allowed

A a = new B(); // yes allowed, where a is a reference of type abstract A but holds B’s object.

@override

The latest java versions are bringing these annotations..

Java program -> mysql (a table in mysql)

@Entity -> database (EmployeeManagement)

@Table (name = ‘Employee’)

Interfaces, Autoboxing, Wrapper classes, Exceptions

Interfaces are 100% abstract in nature.

Interface can contains variables and methods also.

By default all the variables in interface are public static and final in nature.

And all methods are public abstract in nature.

Interface A{

}

Interface B{

}

Class C implements A, B{

}

Interface A extends B{

}

1. Interfaces are not classes. They are 100% abstract in nature and you cant create objects of interfaces but you can create references of it.
2. We define interface using keyword called interface.
3. The class which uses this interface should implement it. The keyword is implements.
4. Multiple interfaces implementation is allowed.
5. Any class implementing an interface should define all the methods of it, if not the class becomes abstract.

Interfaces, this

Wrapper classes:

Int, float, Boolean

But moment Collections came into picture it started demanding for objects and not for primitive types.

So for every primitive type they bought in a wrapper class.

Int – Integer

Float – Float

Char – Character

Boolean – Boolean

But for numerical types Number is the super class.

// boxing

Integer i = new Integer(10) // i is of wrapper type Integer class.

double d = new Double(40)

int i1 = i; // unboxing

But we don’t have to do all these, the jvm will take care and that is why called Autoboxing.

Integer i2 = 50;

Double d2 = 3.5;

Collections, Threads

Exception handling, I/O, Regex

Exception is that made my program not to execute as expected.

Some accidental event that triggered and disturbed my program.

Amazon website – added things into the cart – selected the payment mode --- should go to the delivery page(This is third party)…

The errors/exceptions details will be in the server logs.

Rather using SOP we will use Logger statements which will write the problems into the server log for the developer to troubleshoot the issue.

Object -> Throwable -> 2 types:

1. Errors – No, errors cant be handled…it can be your memory issue, JVM issue, out of memory, RAM issue…..
2. Exceptions – These can be handled because it can be like ArrayIndexOutOfBoundException, FileNotException, NullPoinerException…

Under Exceptions you have 2 types:

-Runtime exceptions

-Non Runtime exceptions

Checked and Unchecked Exceptions:

Checked exceptions are noted at the compile time.

FilenotFound, SQLException, DBException are all checked exceptions.

Unchecked exceptions: Not at the compile time but at the run time.

ArrayIndexOutofBound, NullPointer , Runtime exceptions..

Runtime are all your unchecked exceptions.

Exception handling:

We handle exceptions using try, catch, finally, throw, throws

Try{

Int I =5

Int x = 5/0;

}catch(ArithmeticException ex){

Ex.  
}

Try{

File f = new file()

}catch(FileNotFoundException ex){

}

finally{

}

Try{

}

Catch(){

}

Catch(){

}

Its not a good practice to catch the super type.

Let your exception type be precise.

You can create custom exception class using example: BankException extends Exception

abc 25 20k