JIT convert byte to machine code.

Jre provides runtime environment to the program

Jdk provide Jre as well as development kit

It is mandatory for same name of class file and class in public specifier but not in default

byte 1 -128 to 127

int 4 -2^31 to 2^31-1

short 2 -2^15 TO 2^15-1

long 8 -2^63 to 2^63-1

In java methods are by default are default and virtual in nature.

Virtual : should override

Abstract : must override

Object in java

Abc a; //not object

Abc a = new Abc();

Object in C++

Abc a; //static

Abc \*a = new Abc(); //dynamic

== compare the address

equals() compare the value

which is repeated code is **boiler plate code**

**Spaghetti code** is a pejorative phrase for unstructured and difficult-to-maintain source code.

**class :** a blueprint that describe state (member variables) and implementations of behavior

**object :** instance of class that maintains its state in variables and implements its behavior with methods

Data Hiding: make data member private and cannot access data member directly through object of a that class for that we need getter and setters.\

Abstraction : Access modifier/specifiers

Encapsulation : class/package

Polymorphism: Overloading and Overriding

Inheritance

System : Class

Out : reference of PrintStream

Println : PrintStream Class methods

Pre-written Code:

Library : consist of class files

Framework : multiple library

Component :

Bean :

Services:

1.Public

2.Protected : cannot be called outside class directly but by extending that class can be called.

3.default

4.Private

Method Signature

1.Number of Parameter

2.Type of Parameter

3.Sequence of Parameter

Overloading can be stop using generics

Overloading: Early / Static / compiletime Binding

same method name with different signature.

works can work with or without inheritance.

return type is not considered.

Yes, we can overload

Overriding : Late / Dynamic/ runtime binding

works in inheritance

same method with same signature

can be given same or above access

Overriding can be stop using final keyword

Acces can be same or higher

**Note: If Both Parent and child method(same name and signature) are non-static then it is overriding but if both are static then methods will be called via reference variable but both has to be either static or non static if not you will receive error.**

Abstract class can have all type of member including abstract.

Final : final(not inherit) and abstract(must inherit) cannot be used together in class

Constructor is a special methods used to initialized the member of class.

* Constructor has same name as of the class.
* No return type
* Automatically called whenever object is created.
* If we don’t create any constructor of class then compiler generate default constructor otherwise we have to create.

Super() is used to call parent constructor

This() is used to call its own constructor.

Default constructor

* Child(){super();}

No Argument Constructor

* Child(){}

Interface

Abstract nature

By default Interface have Public Access but it is default in class.

By default variable are public static final

By default methods are public.

In multilevel we have to make class abstract if it is not overriding all the method and then next child class can implement

Since java 8 we have method body inside interface.

JODA api…Date and time api

* Try with resource

Resource in try should be from Closeable interface if we don’t use finally block.

try(Scanner s = new Scanner(System.in)){}

**Syntactic sugar** is **syntax** within a programming language that is designed to make things easier to read or to express. It makes the language "sweeter" for human use: things can be expressed more clearly, more concisely, or in an alternative style that some may prefer.

Static:

Static variable maintains the copy for the class and non-static (instance/member) variable maintain copy for object.

Class Sta{

Static int a;

Static{

Sysout(“static block”);

}

}

Class mainmatic{

Psvm(string[]arg)

{sop(“in main”);

Sop(S.a);}}

-------------static context -------

1.static variable:

2.Static method :

> Non static variable cannot be referenced from static context but vice-versa is possible

> Static variable can be referenced from static as well as non-static context

> We can use non static variable if we create object for example: main();

Java Featured

Enhanced / advanced for loop ; Scanner ;Queue(collection) 1.5

Try with resource 1.7

Lamda/Functional Interface/stream/dateandtimeapi 1.8

Jshell / jdk is modularized / 1.9

Inner Class.

1. **Static inner class with satic mathod**

class Outer {

   static class Inner {

      static public *void* disp() {

         System.out.println("This is an inner class");

      }

   }

1. **Static inner class with non static method**

class Outer {

static class Inner {

public *void* disp() {

System.out.println("This is an inner class");

}

}

}

class inner {

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

/\* Outer.Inner.disp(); \*/

// d:\KACHRA\JavaWork\src\JavaRevise>javac inner.java

inner.java:23: error: non-static method disp() cannot be referenced from a static context

Outer.Inner.disp();

^

1 error

Outer.Inner oi = new Outer.Inner();

oi.disp();

}

}

1. **Non static inner class with static method**
   * Not possible

class Outer {

class Inner {

static public *void* disp() {

System.out.println("This is an inner class");

}

}

/\* void display() {

Inner inner = new Inner();

inner.disp();

} \*/

}

class inner {

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

/\* //Outer.Inner.disp(); \*/

Outer.Inner o = new Outer.Inner();

o.disp();

}

}

Error

d:\KACHRA\JavaWork\src\JavaRevise>javac inner.java

inner.java:3: error: Illegal static declaration in inner class Outer. Inner

**4 . Non static inner class with non static method**

class Outer {

class Inner {

public *void* disp() {

System.out.println("This is an inner class");

}

}

/\* void display() {

Inner inner = new Inner();

inner.disp();

} \*/

}

class inner {

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

/\* //Outer.Inner.disp(); \*/

*Outer* o = new Outer();

Outer.Inner i =o.new Inner();

i.disp();

}

}

OR

class Outer {

class Inner {

public *void* disp() {

System.out.println("This is an inner class");

}

}

*void* display() {

Inner inner = new Inner();

inner.disp();

}

}

class inner {

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

/\* //Outer.Inner.disp(); \*/

*Outer* o = new Outer();

o.display();

}

}

***Annonymous class***

abstract class Cloth{

abstract *void* price();

}

//anonymous inner class

class anonymous{

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

*Cloth* i = new Cloth(){ //i hold the reference of anonymous class

*void* price(){

System.out.println("price:100");

}

};

i.price();

}

}

Or

interface Cloth{

*void* price();

}

//anonymous inner class

class anonymous{

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

*Cloth* i = new Cloth(){ //i hold the reference of anonymous class

public *void* price(){

System.out.println("price:100");

}

};

i.price();

}

}

Interface Java8

* We cannot call static method of interface from object of implemented class

Checked Exception: checked at compile time and throw at runtime. If it is not handled then it must be caught or declared to be thrown

Unchecked Exception: not checked at compile time and throw at runtime.

We have to write min code inside try block because when exception occur the rest of code after exception will not be executed.

Array

Class name for array

[I for integer

[B for Byte

[Z for Boolean

Size of

Array ……length

String ………length()

Collection………. size()

Generic came in 1.5

Collection:

Legacy class : Vector(1.0) but redefined at (1.2) and Stack(1.0)

Reference type to value……………… unboxing

Value type to reference……………… boxing

Insertion Order Preserve :

ArrayList

Ordered:

**Array List :**

Represented in the form of array

By default is 10 memory allocate

Next Allocation Formula: (current capacity\*3/2)+1

**Linked list :**

Memory utilization is better but retrieval is time consuming

**Vector :**

initial capacity : 10 using capacity().

Obect collection

ArrayList ar= new ArrayList<Integer>();

ArrayList<Integer> ar= new ArrayList();

Generic collection

ArrayList<Integer> ar = new ArrayList<>();

ArrayList<Integer> ar = new ArrayList<Integer>();

Iterable…..forEach since 1.8

Function…Consumer

STREAMS:

Processing the Collection and Getting the another transform collection.