**Running Notes for Core Java**

**Day 1**

Programming language contains or Programming language elements

1. **Keywords:** inbuilt and defined identified words to give the definition to the programming language. Each keyword definition is already defined in the programming language by the programming language developers. Compilers or run time environment can understand only the defined words in the programming language.
   1. In Java each keywords represents in lower case. For e.g. class, public, static, void etc.
2. **Identifier:** Names identified to the variable, packages, Interface etc called as Identifier, Whatever names given by the developer is called identifier. There are some rules for Identifier
   1. No identifier can’t be chosen as keywords.

**Note: No can’t be chosen at the first place and it’s allowed from the second word onwards**

* 1. (A or a, \_, $), First letter of identifier should be any alphabet (lower or upper), underscore(\_) or dollar($).

$ and lower case is discouraged

**Note: only two Special characters are allowed only \_ and $ but space ( ) is not allowed in any identifier**

* 1. From second alphabet onwards, developer can choose alphabet (upper/lower), underscore, dollar and digits are allowed.

**Valid Identifier**: e.g: Hello, Test1, Hello\_to\_all, $Manager

**Invalid Identifier**: static, void, 1test, Hello all, Manager%toall (special character also can’t used)

1. **Literal**: Any value considered as a literal, there different type of values exist like
   1. **int literal:** 10, 50, 26, 0, -20 etc
   2. **double:** 1.5, 2.8, 0.8, -2.8 , -5.6
   3. **char:** any character passed from the keyboard, ‘a’, ‘$’, ‘P’, ‘e’, ‘F’, ‘9’, ‘#’, always should be single quote.
   4. **Boolean:** true or false
   5. **String:** string value, under double quotes considered as a string, and even space and number also ”Hello to all”, “786”, “@#$%&testToAll57”
   6. First 4 type(int, double, char, boolean) of literal considered a primitive and String is derived literal
2. **Built-in:** Frequently used or common features which is already developed from the programming language developers.

**Note:** In Java almost 60% is inbuilt so just we to learn how to use these inbuilt features, for e.g. collectionAPI, Threads, String, Wrapper classes etc, Sytem,

1. **Ctrl + J** can be used in Editplus to copy the line

Practical for today

Expression results like mathematical results also can be printed inside println method, Boolean expression also can be come inside println method

**Day 2**

* Declaration(one time), initialization(any number of time), Usage(any number of time)
* int i = 10, While declaring a variable also value can be initialize to the variable
* variable can be use or reinitialize any number of time in the program but declare only one time, While declaring developer has to give data type
* Any number of variable can be declared and initialized in the program
* Java is a case sensitive
* + operator is used to add anything to the string.
* Inside double quote whatever is there will be printing
* Moving variable to string or string to variable, use plus(+) operator
* Same data type variable can be declare in the same line like int I, j, k, m;
* Variables can be declare and initialize in the same line of same type

Local Variables

* There is no condition like if any variable is declared and initialized then local variable has to use. Means not necessary that every variable’ should be in use. But it’s not recommended as per the coding standard
* Local variable should not use without initialization else we will get some compilation error
* Every local variable should be initialize before usage
* Please analyze all the compilation errors very properly
* If any compilation error in the program then there wouldn’t be any corresponding class file
* Value assignment is also kind of usage like j = i (if i is not initialized)
* No local variable can be used without initialization
* In one method more than one variable should not have same identifier for e.g. int i =10 ; int i = 20;

(variable i is already defined in method main(String[]))

* Two variable should not have same name in the same method is not allowed
* System.out.println(); simply printing and come to new line
* System.out.print(); print method cannot be used without any argument(compile time error), println can be used without any argument because it is having new line character.

**Day 3**

Unary operator:

One unit effect increment or decrement to the variable is called unary operator. Unary operator applied only on variables not on any digit etc. The order of execution is always Left 🡪 Right in Java

* Two type of unary operators are available:
  + **++ operator:** which is used to increment variable value by one
  + **-- operator:** which is used to decrement variable value by one
* Both unary operator can be used in two different ways
  + **Post usage:** First variable then operator means first usage then increments, changed value should not effect at the place of operation and it should effect to the next usage. Prog: app2A
  + **Pre usage:** First operator then variable means first increment/decrement then incremented or decremented value in usage. changed value should effect at the place of operation the next usage.

**Day 4**

* i = i++; // after increment as i = 1 but left side value assigned to 0 so finally i value 1 will be replaced with 0

Temporary variable having value 0 and that will assign to left

* i = ++i; // no issue here
* Pre gives more performance because pre doesn’t use any temporary variable internally
* .. represent to going one step back

**Day 5**

**if-Blocks**

* If (true){}, if(false), if(!false), if(!true)
* boolean b1 = true; if(b1)
* boolean b1 = false; if(b1 = true) // we can modify the variable while using , b1 got modified here to true
* if(i == 10)
* if(i = 10) : compile time error, because if block always needs Boolean type with true or false. incompatible types: int cannot be converted to Boolean
* if block variable can’t be used outside of if block: compile time error
* if there is no opening and closing braces for if block then only one statement comes just after if will be under conditional block, other’s will execute
* if(true || false) // OR operator, at least one has to true (pass), any rule should be passed
* if(true && false) // AND operator, both should be passed, all rules should be passes
* if(i++ == 1 || i++ == 1), fist condition will be checked if true then with OR operator next condition will not be checked but if first is false then the second will be checked.
* With AND just vice versa, first condition will be checked if first is true
* if(false); in this condition if block is already ended becz of ; so each & every line will be executed means just after the if statement also line will be executed .

**else-Blocks**

* In the case of if block will be executing then else block will not be executed, else block always need if block(error :else without if)
* Immediately after “if” should have else block otherwise compiler will generate error, so in between if and else block should not keep any statements in the middle
* Opening and closing braces are not mandatory but if so then just after “if” or “else” the statement comes under conditional .
* if(true)

if(false) 🡺 this statement is the part of above if block, if(false) -> [ if(true) -> sop}, so here if(true) will not be executed.

Sop(“”)

* else block is always belongs to nearest if block so

if(false) //

if(true)

sop

else

sop

sop

* if one “else if(true)” is executing then other “else if(true)” block will not be executing

**Day 6**

**Switch case**

* Case can have any integer representation value
* For last case break is not required
* if no break statement then control will jump to the next cases and will be executing all the cases one by one
* if no case matched then default will be executing
* if case matched then default will not be executing
* default case can be anywhere inside switch case or in the middle
* while keeping a default somewhere in the middle then break is required
* if default at the end then break is not required
* In one switch case, only one default is allowed else CTE error: duplicate default label
* switch(5): Such statements also possible
* General use case: if more than statement having the same solution then we can use case statements without break because we are looking for a common solution.
* Character also can be passed to the switch case because every character having a unique ASCII code, ASCII code is always a integer representation
* Switch case always required int type
* In Java to represent any number without decimal 4 data types are there , increasing range given below

Byte🡪 short🡪 int🡪 long

* + Byte (auto convert to int, it’s narrower to int)
  + short (auto convert to int, it’s narrower to int)
  + int
  + long ( auto convert to int is not possible because it’s wider to int, in this case type cast is required )
* long datatype can’t passed to the switch case
* every case value should be in the range of datatype which is passed as a parameter to the switch case

error CTE: incompatible types: possible lossy conversion from int to byte

* **Data types range**
  + byte: System.out.println("byte-min: " +Byte.MIN\_VALUE); 🡺 -128
  + System.out.println("byte-max: " +Byte.MAX\_VALUE);
  + System.out.println("Short-min: " +Short.MIN\_VALUE);
  + System.out.println("Short-max: " +Short.MAX\_VALUE);
  + System.out.println("Integer-min: " +Integer.MIN\_VALUE);
  + System.out.println("Integer-max: " +Integer.MAX\_VALUE);
  + System.out.println("Long-min: " +Long.MIN\_VALUE);
  + System.out.println("Long-max: " +Long.MAX\_VALUE);
* Final variable can’t be change further
* Switch argument always can be any kind of literal or final variable but case j: is not possible because case always need constant values, in the case of case j: CTE error: error: constant expression required
* final variable treated as a perfect constant so can be used in the cases
* Final variable will be treated as a perfect constant if it’s initialized while declaring and only those kind of final variable can be used in the switch cases.

Day 7:

LOOPS

* for(int i = 1; i<=10; i++) // loop header having 3 section
  + initialization block
  + loop condition should return true
  + changes
* inside for loop body we can put n no of statements
* I is local to for loop becz initialized inside the for loop so it can’t be used outside of for block
* Initialization block execute only once
* i++ at first will not be executing from the next time onwards will be executing
* at first only initialization and condition will be executing
* from second time execution: changes then condition then body
* i = i-2 🡺 i -=2
* if not keeping any opening and closing braces then only one statement would the body of the for loop

and body will be executing for all the change section

* declaration not initialization make the things local
* if for loop ended with ; then no body of for loop and other statements will be executing doesn’t matter of curly braces

**Day 8**

**Continue:** if continue is there in the loop then after than other statement of the body will not be executing, and continue always should be last statement of the block doesn’t matter if, else, while or for etc, by default continue is for the current loop and it will continue the next iteration of the current loop

**Break:** once break will be executing entire loop will be skipped, if we want to terminate the loop explicitly , and break also should be as a last statement of the block doesn’t matter if, else, while or for etc, by default break is for the current loop and it will break the current loop

**Initialization Block: for(int i = 0, j = 20; i < 10; i++, j--)🡺** Under: Multiple same type of variable can be initialized using comma as separator, and also any no of statement using comma can be possible even sop also possible

**Conditional block:** multiple conditional statement can be used but using && or || symble.

**Changes Block:** Any no of changes can be possible by using comma as a separator , , and also any no of statement using comma can be possible even sop also possible

for(int i = 1, System.out.println("Initialization"): CTE , if we are declaring a variable in the initialization block then we can’t keep multiple statements because compiler expect other variable to be declare not any other statements. Don’t initialize variable in the for loop header

we can give the name to the loop (Identifier:) for e.g. loop1: on the top of the loop and with this we can reach to outer loop if we want otherwise by default inner loop always so somehow we can control outer loop from the inner loop then outer loop should have the name , directly will go to the next iteration

**Day 9**

While-loop:

* While loop header Only one section , Boolean section then after that body and inside body changes
* Declaration of variable before Boolean section no separate section inside loop header
* Structure wise and for and while loop are different but flow wise it same
* For loop is somehow independent fro while loop is dependent becz it’s depends on outsite variable declaration, in while loop within body increment/decrement is possible
* In while and for loop, for all the iteration including first iteration Boolean condition should pass.
* After continue no other statements will be executing , even increament/decrement will not be executing, if changes are there after continue then infinite loop
* With break also change should be after or before the condition
* Inside of any loop any loop can be possible
* Be default break will break to the current loop and the outer loop if any
* Outside of while loop we can give any label (valid java identifier)[controlling outer loop from inner loop]

Do while:

* Without condition control going inside the body, First body then condition will be checked
* Three points to remember in the case of do-while
  + Minimum one iteration is guarantied
  + Do while should end with semicolon(;)
  + Vaiable used in Boolean condition of do while that should be declare before do while not inside the body of do while if we will declare in the body of do while then this variable we can’t use as Boolean variable
* If there is no opening and closing braces for while loop then just next statement will be iteratating
* while (i > 0); 🡺 infinite loop [and the same for loop]

Day 10: Revision class

Download & Installation of Java

Oracle.com/index.html 🡪 Sign in/Register 🡪 mouse over Downloads 🡪 click on Java for developers 🡪 major version and update no(1.8)🡪 click on Java Icon 🡪 Accept🡪 x86(32 bit) x64(64bit) 🡪 download x86 becz it can run on 64 bit as well🡪

Java Platform independency: Developed code can be run in any environment but JVM is varying is from one OS to different OS that why JDK is not compatible or platform independent

Better should have 4 drives, E drives -> development purpose, F drives: Backup, D drive🡪 software installation , C drive OS

While Installating java: don’t install public jre 🡪 not for development purpose it’s for production purpose so remove public JRE, Under development tools JRE also available and that used for code development

Install Java under JDK8.0 folder.

Setting path environment variable

* Javac -version, and Java –version, go to bin folder of jdk8.0 and run the commands,
* Command prompt wise: <not advisable >
  + Set PATH=<jdk bin folder location> [Specific to the command prompt]
  + You can use any slash, either forward or backward
* Go my computer🡪 advance settings🡪 environment variable🡪 user section(specific users)[recommended] or System section(for all the users )🡪 user variable , variable name: path, variable value: <jdk bin folder location >
* Echo %PATH% 🡪 can see all the value of the path environment variable

Programming elements 🡺 Go back to Day 1 above from here

* Keywords
* Literal
* Identifier
* Build-in

How to download Editplus: Go to editplus.com/download.html 🡪 download

-d 🡺 where classes files will be stored

.. 🡪 Previous

Some old points to remember

* Local variable should not use without initialize in Java but in C compiler will be suceesss
* Oder: first declaration and then usage
* In one method, two local variable should not have the same name. CTE
* Under if always Boolean value, if(i)//CTE
* Under if and or , if first is false then only 2nd condition will be checked
* With && operator if first one is false then other one will not be executing
* And remember i incremented values will be used always
* if(false)

if(true)

sop()

sop() // only last statement will be executing because inner if is not executing and inner if containing one statement. If you will curly braces then would be very clear

int x = 20

* if((x=30)) so x is not modified in the body of if and it will be change permanently

**app8 and app9:**

**Global members: members of the class, attribute, methods**

Attribute are in different – 2 datatypes, every global variable automatically initialize to it’s default value and it’s depends on it’s data type, like int 0 , float 0.0, Boolean flase, String null

* **Global Static variables**
  + Global variables default values is 0 and if not initializing but local variable we have to initialize explicitly
  + Two global and local variable should not have the same name: CTE
  + One global and one local variable with the same name: it is possible but local one will get the preferences.
  + Global variable also can be initialize while declaring and called initializer
  + static int x = 10; static boolean x = true; 🡺 CTE
  + To access global variable if any local variable name is same as global variable🡺 ClassName.variableName,
  + better global always use with class name
  + Local members are the part of the method
  + Static can’t be to the local members, static always are the members of class
* **Global Static methods**
  + Java command always will run main method by default not any other methods
  + No matter method or statement is executing or not javac command will check every statement is syntactically correct or not. If So CTE
  + From main method only we can call to other method then flow can go ….
  + In one class any no of method can be incorporated
  + Non static means completely object oriented programming
  + Main method always should be public and other method are optional
  + Defined Method can be call at any time and inside method any no of statements can be possible
  + Using methods we can achieve the reusability
  + Local variable of one method cannot be used to any other method // CTE
  + In different methods same name variable is absolutely possible
  + Global variable can be used in any methods throughout the current class
  + Keep in mind global variable values can be change to the local variable also so take care and keep the difference b/w local and global
  + In general every method should have a return type and this return type can be void or primitive(int, char, Boolean, double) or derived datatype(string)
  + Return type should be always before the name of the method like void test(), void main()
  + Static method and variable can be use without object and using the class name

**Void return type**

* + Void means should not return any value
  + Hiding ???
  + Main method is void because it will not return any value ever and it’s static so no need of any objects as well.
  + One CTE: unexpected return value
  + Return statement is optional if return type of the method is void and it’s allowed
  + Inside non static block can we use static ????
  + Return statement should always be the last statement of the current method, very similar to break and continue else CTE: Unreachable statement. (break, continue, return, throw should be last statement always in the current block)
  + \*\*\*Requirement: Not required to continuing further the we can use return then after return all the statement in the current method will not be executing, e.g. inside main method under if block return then after return main method will exit
  + Break and continue is for loops and return is for methods

**Primitive return type:**

* Any data type which is already defined in the PL is called Primitive data type, In Java 8 primitive data types are available:

byte, short, int, long (representing no without decimal)

Float, double (representing no with decimal, based on range we can choose float or double)

boolean (true/false)

char (single char)

* If method return type other than void then return statement with value is mandatory , CTE: missing return statement
* For method just take care, it should return something if returntype is other than void
* Assignment always should be homogeneous means both side should be same return type
* System.out.println(test()); is possible so whatever will return from test() method will print and such things can be used for debugging the code and it would be very feasible
* CTE: non-static method test() cannot be referenced from a static context

**Derived Data type**

* Any class , interface and enum are the drived datatype , String also comes under drived data type
* Method also can have drived data type return type,
* While checking method we have to take care : static calling to non static etc, method return type and homogeneous assignment

**Method with arguments:**

* Method can take any no of arguments by using comma as a separator e.g. test(int i, boolean j )
* Arguments of the method are local to the same method
* Arguments are assigning automatically from the calling statements
* Arguments can be used straight-away without initialization
* The value passed from the calling statement should have same datetype in the method definition
* Where global variable is declaring and at the same place if initializing then it’s called initializer
* With initializer necessary to use regular reference
* Illegal forward reference CTE: if no proper regular reference for the initializers : if using global variable as initializers then we have to use regular reference
* Method works with forward reference and it’s not illegal
* Forward reference means defined first then usage
* We can use any method in the initialize is assignment is proper
* Global variable only in initialize check for regular or forward reference , and should not use any variable with forward reference
* Method can be used anywhere as long as data types are same
* Method return type and returned value data type should be same
* In assignment and in method always both side data type will be checked
* **Loading procedure:**
  + Class will load first
  + And from the class only all the static members will be loading in the memory and while loading all that static member will be loading with it’s default values
  + All the initializer will be executing from top to bottom
  + Class completely loaded now to the memory
  + All the initializers calls before the main method and this way we can call any method before main method
* **Static initialization block(SIB):**  SIB also comes under initializers so while executing initializers SIB also will be executing
  + Inspite of separate method in the initialization we can go for SIBs
  + Under sib x = 23 is correct because it’s just an assignment not the usage but with usage always forward reference comes in to the picture
  + **y = x;**  this is also kind of usage because using x to provide a value to y
  + We can call main explicitly and to main method we can pass string array or null for example: main(null); Note: all drived data type default value is null
  + SIBs are executing once class is loading to the memory
  + Static int i ; is the initializer
* **Developing multiple classes** 
  + public class: if there is a public keyword in front of the class
  + if class is a public then file name should be the public class name
  + if class is a non public then file name can be chooses as any name like hello.java
  + In one java file more than one class can’t be the public
  + If we want to use one class static members to any other class then we can use by using class name .
  + Main method class only we have to run
  + For saving we have to see the public class not the thing that class should have main method
  + While running compiler will make the class file same as the class names
  + Remember static members load only once for the one execution
  + Whichever class is calling the same one will load
  + Any class is loading to the memory only one time and once class is loading then only static members will be loading
  + One class of one java file can be used to another java file into different class
  + **Multiple java file can be compile together by using space a separator but while running we have to run each class separately there is no way to run multiple classes together**
  + **java -cp ../classes C**
  + other option to compile multiple java files together: \*.java

**App11\_**

* **Depending classes developed in different files**
  + Class D is dependent on class E, so for compilation we have to use the class where we have the main class developed , compilation will be success if are the file d and e are there in the same folder else CTE for running run the main class file
  + Main method args can be passed to call other class main method if needed
  + If we want to use one class from one file to different class of different file then we have to remember that file name should same as the class name so that it can compile successfully that’s why always we have to save files same as the class name. but we have such exception scenerion then we have to compile all the files separately(space separator we can use)
  + **Industry: one file should have one class only**
  + **In one file can have more than one classes but don’t use it**
  + **Industry: multiple classes and each should be in separate files**
* **CommandLine Argument**
  + Main method argument is args and length is a attribute, property or field
    - **System.out.println(args.length);** this way we are passing argunment to args array and getting the length of the array , in C++ class name also considered as argument but in Java class name is not considered as argument.
  + if args is a null then we will get null pointor exception
  + if args is emply then we will get 0
  + all the data type we can pass as a command line argument to the main method (because all the data type auto converting to String and storing to string array)
  + **int i = Integer.parseInt(s1);**
  + **double j = Double.parseDouble(s2);**
  + all the argument passed from the command line will be in the String form so we have to convert to exact desired format
  + **Input at the run time: import java.util.Scanner;** **Scanner sc = new Scanner(System.in); String s1 = sc.next();**
  + **System.out is for writing**
* **Enhanced for loop or (introduced in jdk1.5)**
  + for each loop: int[] x = {10, 20, 1, 5, 80, 25}; for(int i: x) : here loop will execute for each element of the array

**app12\_: Logical coding**

* 21 to 42 hw

App11: Basics for Encapsulations

* Non static global variable can’t be used straightway inside static block : CTE: non static can’t be referenced to static context
* Object creation: Using new operator possible, <classname> <objectName> = new classname(); for example G g1 = new G()🡺 Object type is G so reference variable data type also should be G type
* Non static members can be used to static context by using only reference variable
* When class in loading all static members are loading but non static are not loading but while creating an object of the class all the non static members will be loading.
* Globle variable static or non static default value is 0
* Any any variable type is a class type then it’s drived data type
* Primitive variable should not be assigned to objects
* Object is creating to the class, object type would be class type, and using reference variable only (class type ) object can be used
* Compiler will be checking only syntax is correct or not and compile will be success.
* Empty will also can be compile but .class file will not be creating because class files will be generating corresponding to each class in the java file.
* One object can have more than one reference variable and using any ref variable value of the variable can be change
* Any one reference variable can change the value of the variable
* As one object can have more than one object reference so these reference can have to or from different methods
* Object can not be moved from one place to other place but only reference variable only can be moved but take care the where ref variable is moving and there class type of the target variable
* Object can be created in any method and ref of that object we can pass to anywhere
* One object can be assign to other object and after assigning both object will refered to the same location \*\*\*\*
* One block object will be limited to the specific block
* For one class we can create any no of objects\*\*\*
* Static only one time will be loading to the memory once class will load(one time for the entire execution) and Non Static will load to the memory for each object creation. Static execution dependent and non static object dependent.
* For each object variable will have the separate copy in the momory
* Once object is created in the momory then it can be used anywhere in the complete class or program or project using or passing reference variable .
* this: is the keyword and it’s available only for non static members
* print object: class@memoryaddress
* which reference you are using to call a method then same reference will behave like a this inside the method.
* This is not available ot statis methosde

**Constructor:**

* Constructor name should have same as the class name.
* Constructor should not have any return type
* Constructor is executing automatically when object is creating
* Cons we can use as non static initialize block so it’s known as non static initializer
* Through constructor we can initialize non static members
* E e1 = new E(9); is possible , constructor can take any no of argument same like methods , cons(int x) but we have to pass the argument while creating an object.
* In one class any no of constructor can be created
* In one class any no of constructor can be created with arguments or no args
* By using constructor any no of objects can be created
* There is no condition like if we have defined constructor or method then we have to use, it’s purely optional
* J(int p, int q): called signature of the constructor, As long as signature is changing we can develop any no constructors

**Constructor overloading :** Developing multiple constructors in a Class by changing the signatures of the constructors is called constructor overloading , should have different data type and no of args different

* CTE: duplicate can be identified in case of constructor throw constructor signature
* Constructor creation is an optional because if we are not creating any constructor in the class then compiler will create default constructor and this will no arg constructor tht wy we will not get any CTE if no constructor in the class
* While object is creating we have to use available constructor
* If we are passing any arg while creating an object then we should have that corresponding cons definition in the class else CTE
* If class doesn’t hv any cons then only compiler will be generating no arg constructor, if already there is any constructor in the class then compiler will not be providing any no arg constructor
* Static members can be used straightaway without class name in the same class
* Static members can be used outside of the class using class name.
* If we are calling static members using reference variable then compiler will be replacing reference variable to the class name.
* So overall we can say static member can be used outside of the class using class name or ref variable
* Static members are the common to each object (each object ref variable will be coverting to the class name so if any ref variable will change the value of the static member then it’s same for all the objects.)
* If And data members (variable and method) is common or global to the class or each objects then we have use static variable and static method
* Non static members are not common to each object and it’s specific to each object. For each time once object is created separate copy of constructor will be loaded into the memory.
* To find out the no of objects we have to increment static counter in each constructor
* this(); will call to the no arg constructor and it always should be the first statement of the constructor body
* this(10); possible but always should be the first statement
* if we want multiple constructor for one object creation then we can go for this
* For every this calling statement there should have corresponding constructor in the class or there should have suitable definition block
* With “**this**” calling statement in the case of contructors, recursion invocation is not allowed doesn’t matter we are creating an object or not
* For each calling should have perfect definition block
* Constructor we can’t call by it’s name for ex: Z(); we have to use this keyword only no other way. In such cases constructor will look for method
* Constructor should not have a return type that’s how compiler identify that it’s constructor
* Constructor can be called only in two different ways
  + By creating an objects throw new operator
  + Using **“this”** calling statement

**App14:**

**IIB**

* IIB will be calling for each object creation , no matter which object we are creating
* If any initialization is needed object wise then use IIB because for each object creation IIB will be called and initialize the value
* If any constructor independent initialization then use IIB
* SIB are executing when class is loading , all SIB are executing from top to bottom while class is loading
* ALL IIB’s are executing from top to bottom when object is creating , one class can have any no of IIBs, first IIB then constructor will be executing
* IIB execution is object wise not the constructor wise

**Encapsulation**

Entities can be represented by the class, entity can have properties and behavior, how to represent entity property and behavior by attributes (represent properties) and fields (represent method) of the class. Behavior can be representing by methods. Designing a class is nothing but encapsulation, encapsulate or group one entity information into one group means class is called encapsulation. Class members are attributes. If something is global (common to each object) then make static.

Note: class is the blueprint of each similar type of object

Note: constructor, IIB and SIB used for initialization as per the requirement, When we need common initialization for all the objects in the class then we can go for IIB and if we need initialization object wise then go for constructor

Note: constructor is also called as initializer

Using encapsulation we are not mixing one entity to another entity, all entities would be separated using encapsulation

**Inheritance**

* extends keyword: After encapsulating one entity into one class we can extends to another entity

A

B

Super class, all members of A inheriting to class B

Sub class, In class B all the members of class A are available

* Every member of super class inheriting to sub class, so this way we are achieving the reusability throw inheritance
* SIB, constructors and IIB will not inheriting to the sub class
* Super class initializes is not inheriting to the sub class, constructor are not surviving in the inheritance
* If any constructors are there in the sub class then compiler will be checking the first statement in the constructor body and found if any this or super statement is there or not if nothing is there then compiler will be keep one super with no arg statement inside the constructor body and super class no arg constructor will be called.
* Every class should have minimum one constructor if deverloper are not keeping then compiler auto will keep
* If any class doesn’t have any extends keywrd then compilers will keep one extends statement like **extends Object** so this way it will extend object class, object is built in class
* If no this and super then compiler will keep only super with no args
* Every class is the sub class of object class

**Constructor with super and IIB**

* Inside constructor body first super then IIB then sop
  1. Super
  2. IIB
  3. Constructor body
* Remember super is always refer to the parent class always
* IIB will be executed only one time for one object creation.
* IF this keyword is there then first IIB will be called
* Inside of any constructor if “this” statement is there then no super call statement will incorporate
* Any constructor having this then no IIB inside that constructor

**SIB involvement with Inheritance**

* First super class will be loading then sub class will be loading to the memory
* First super class will be loading with SIBs then object creation and while creating object IIB will be called as per super and this rules

App 17:

class A

{

int i;

}

class B

{

A obj; //attribute of type A = has a relation, ref variable = 0

public static void main(String[] args)

{

B b1 = new B();

b1.obj = new A();

b1.obj.i = 20;

System.out.println(b1.obj.i);

}

}

b1

I = 0 , 20

B object

should not all any member with null object referenc

F is a whole and E is a part, inside E attribute, we can’t use E part attribute to the F whole , need one condition can be fulfilled

Two type of has a relationship

* 1. Aggregation
  2. Composition

Aggregation: Whenever whole is removing or destroying still part is surviving is called aggregation. For example, car has a driver even if car is destroyed still driver can survive , not a tight bound between whole and part

Composition: Whenever whole is destroying then part will also be destroying is called composition for example care has a engine so if engine destroyed then car also destroyed , company and website, a very tight bound between whole and part

IS A relationship

Inheritance fulfilled the condition of is a relationship because all the members of super class are available to the sub class without following any condition

m1

n1

q1

**app18: Packages**

* Package statement should be the first statement in the java file
* From src only we have to trigger javac command but we have to specify packagename/A.java and while running also we have to specify package name like pack1.A
* Compilation: pack1/A.java (use forward slash)
* Running: pack1.A (use dot operator)
* Package inside package can be developed
* Package contain java files and another packages
* In case of packages, we should not develop any folder with naming as capital letters, only software installation folder can have first letter as capital
* Every java file should be assigned to any package because we can separate one type of java files into one folders
* Naming collision can be avoided means in one folder not possible to have two files with same name so in different folders we can keep same naming conventions java files (same name to multiple classes)
* One package members can be used straight away to and from the another members , no need to import anything , current package members can be used straight away without following any condition.
* One package members can’t be used to another package members straight away, we have to follow to two conditions
  + Public members only can be used in another package
  + While using other package members into different package then use along with the package name or import
* By default compiler will search classes in the current package not in the another package
* By accessing we can use with package name like pack1.A.main(arga) or pack1.A.i but A should be public else we cannot use, **only public members only can be accessed to the other packages**
* Import always should be after package statement if any
* If class is public then it’s scoped is to the current package so better always make class as public
* While importing we have to specify which particular members we want to use in different package or else use import pack1.\*; by this all the public members we can use

**Access levels**

There are 4 access levels in Java

* Private
* (default without specifying anything)
* Protected
* Public

Note: IIB and SIB should not declare with any access levels.

**PRIVATE:**

**Usage:** it should be used only for the member of the class not for the class itself; class members can be declared as private for example attribute, methods, constructors,. So all these can be declared as private. Private is a keyword. If any attribute or method or constructor is a private then it should be used within the current class. Private members limited to usage within the current class we cannot used outside of the class. For example private int temp then it cannot be used outside of the class. Class cannot be a private but only its members except sib and iib.

**Inheritance:** Private members are not inherited to the sub class.

**--------------- (default access level):**

* Any Class can be default and its members also can be default scope.
* If class is default then this class can be used only within the current package not outside of the package.
* If any member of the class is declared as default then that members can be used in the current class and also other classes of the same package.
* Default scoped members inherited to sub classes of same package.
* Default scope also called as package scope because default members can be used only in the current package.

**Protected**

* **Usage:** Protected members can be used within the current class and also other classes of the same package. In case of usage there is no difference between default and protected.
* **Inheritance:** protected members are inheriting to all the sub classes no matters where this sub class is developed either sub class is developed in the current package or some other package.
* The main goal to develop protected is inheritance.
* Protected cannot be used for the class itself only members of the class can be declared as protected. (similar to private)

**Public:**

* **Usage:** public members can be used anywhere, there is no restriction on usage of public members
* **Inheritance:** Public members can be inherited to any sub class, doesn’t matter where this sub class is available.
* public access level can be used for the class and also to the members of the class

Different between protected and public are same in case of inheritance but there is one difference whenever protected members inherited to the sub class of other package then that members is protected to only to this inherited sub class not other classes. Members can’t be used outside of this class.

With respected to public members can be used in any class of another package.

Different private and default:

Diff between protect and public:

* protected usage with the current package public usage any where
* protected members inherited to the sub class see above

Note: In any of the java file if any public class then we have to save as class name only

private members can not be accessed outside of the current class

default, protected and public members can be access anywhere with in the same package.

If any class is a public then only we use that class in different package by using import else no

Private < default < protected < public

Default as keyword should not be used

If class is containing only private constructor then we cannot create object of that class into the different class

Requirement is how to restrict object creation of one class to another class then we can create private constructors

Every method and constructor should be syntactically correct , if class containing private constructor then we can not develop the sub classes

Private constructors:

* restricting object creation to another class
* restricting sub class creation

Access level of default constructor: default constructor access type is same as class access

E

Pack 2

A

B

Pack 1

D

C

G

F

A is public class, i private, j default , k protected, l public,

C : develop c class, make c sub class as A, create object to c and access members of class A

With A ref variable we can use K inside pack2 but we can use with F ref variable we can use A protected members

Note: IIB and SIB should not declare with any access levels. : Why don’t have access levels : because SIB and IIB don’t have any names and developers are not calling sibs and iibs because they don’t have any names, so no need of access levels

**Abstract;**

Any method unable to define inside a class then we can declare as abstract classs

Void test1() : define method , concrete, implemented method

{

}

Abstract void test2(); undefined method , not implemented, doesn’t have an body, so such method should be declare as with abstract keyword , in class if any one method declared as abstract then class also should be declared as abstract class, for abstract class also .class file will be generating while compiling ,

* We cannot create an object of abstract class because it’s not fully developed but we can create reference variable for the abstract class

For example A a1 = new A(); CTE, A a1 = null ; correct, null can be assigned to any reference variable , null is a keyword which is using to assigning to any reference variable

* We can use abstract class for data type purpose , wherever data type is required there we can use abstract class
* Abstract class can be used for method return type or for argument data type of the method.
* We can develop sub class to abstract class, in the sub class we can implement all the inherited abstract method if any one method is not implemented inside sub class then sub class also we have to declare as abstract class.
* If some method we don’t know how to develop then we can declare as abstract class
* One person only can develop the class diagram or design diagram
* \*\* without abstract we cannot achieve expertise.
* Inheritance is very much required for abstraction
* We cannot create abstract constructors because constructor cannot be inheriting to the sub class
* Abstract keyword cannot be used for variables because variable is used just to keep values and we are not defining anything with variables.
* Abstract keyword also we cannot keep with IIB and SIBs because IIB and SIB also not inheriting to the sub class.
* Inside abstract class we can develop any no of concrete constructors but not abstract constructors , every class should have minimum one constructor doesn’t matter class is a concrete or abstract
* Abstract class constructors are calling when sub class object is creating with the help of super calling statement inside constructors
* Abstract class will also have minimum one constructors because definitely for abstract class will have one concrete class and while creating an object of concrete class then with super abstractor class constructors will be called automatically.
* If class doesn’t having any abstract method then also can we declare class as abstract ? 🡺 Yes, it possible , allowed

Any class can be abstract and developer is allowed to keep abstract method or not so doesn’t matter class is containing abstract method.

* If class doesn’t have any abstract method and still class is abstract then no one can create the object of this abstract class so this way we are achieving object creation restrictness.
* We cannot develop 100% abstract class because we have one default constructor inside and constructor cannot be abstract
* We can develop 100% concrete class and declare as abstract: possible
* In order to achive 100% abstractness , we need interface

For example

abstract class A

{

abstract void test1();

abstract void test2();

}

Abstract class B

{

100% concrete class

}

abstract class A

{

Any no of or one abstract method

}

abstract class A

{

------------------

}

Class B extends A

{

}

**Interface:**

* 100% abstraction can be achieved throw interface.
* Interface can became member of the java file like class A
* For interface also class file is generating by the compiler while compiling the code
* By default interface is an abstract.
* E.g. interface A{ } and abstract interface A{} both are same.
* Inside interface we cannot develop any concrete (defined) members like method with definition.
* Inside interface we cannot develop any iib, sibs, and constructor.
* Inside interface we can develop two type of members
  + Constants
  + Abstract methods
* Interface A { } public interface B { }, by default every interface is abstract
* A is default scope and will be used in the current package, B is a public so it can be used anywhere in other packages also.
* Every members of the interface is a public by default but remember interface is not public but A and B is abstract.
* If class is having abstract method then abstract keyword with class is mandatory.
* Abstract class A { abstract void test(); }
* Interface B { void test(); } // abstract and public keyword is optional for interface method because inside interface all are abstract method by default
* Interface members cannot be declare as private or protected and should not declare as private or protected. By default all the members are public.
* Interface A { void test() ; } class B implements A { public void test() { } }
* We cannot create an object to interface because it’s 100% abstract one.
* We can create a reference variable for interface
* We can use interface for data type purpose, where data type is required we can use interface.
* Interface can be used for method argument data type purpose.
* implements is the keyword which is used
* while implementing interface methods to the sub class we have to use public keyword for the methods because by default all the members of interface are public. Else we will get CTE
* inside B if we are not defining test method then test method qualified as abstract then test method should declare as abstract.
* One class can implement any no of super interfaces like class B implements A, C , D etc
* We can extends interface to interface
* Interface A { } interface B extends A { } //class to class extends , interface to interface again extends but class to interface we have to use implements.
* We cannot develop interface by making use of super class
* Order extends then implements

Practical notes:

* For abstract class also .class file will be generating
* Minimum one abstract method then class will be qualified as abstract class else error class cannot override the abstract method
* Abstract method while declaring should not have braces else we will get CTE because it’s not qualified as abstract method
* In one abstract class any no of abstract and concert method can have
* class is abstract; cannot be instantiated : for abstract class object cannot be created.
* Abstract class can be used for data type and return type purpose.
* Abstract classes are allowed to be used as data type purpose, but what is the use ???
* If class doesn’t having any abstract method then also we can create class as abstract as it’s very optional so this way we are achieving object restriction of this abstract class.
* Is it possible to run abstract class having main method : yes, because main is a static method so for this object creation is not required and this way without object creation also main method will be executing successfully and generally we can’t create object of abstract class.
* Every class (abstract or concert ) will have minimum one constructor, if developer is not keeping then compiler will keep one with no arg constructor.
* Inside concreat class we can not declare any abstract method but vice versa is possible
* When abstract class constructor will be called: while creating any object of the sub class.
* Inside abstract class we can keep IIB and SIB, constructors and methods.
* Abstract class is not 100% abstract
* So to achive 100% abstraction we have to use interfaces

Interface practical:

* Interface also can become a member of the java file and class file also will be generating while compiling
* Abstract keyword is optional for the interface
* Interface doesn’t allow any concreate method and even no constructor , no IIB and no SIBs are allowed inside Interface.
* Inside interface we can keep two type of members
  + Constants
  + Abstract methods.

for example every global variable considered as constant then we can declare inside interface

* int x; CTE inside interface
* every method of interface is abstract by default so need to mention explicitly
* by default every member of interface is public by default so if we put private or protected then it’s compiler time error
* // public keyword is optional here
* Interface itself is not public by default but it’s members are public by default

**App23**

* // by default interface is 100% abstract then we can create an object of an Interface
* // interface can be used for datatype purpose, wherever data type is require we can use interface
* // method argument type, reference variable type we can use interface type
* // be default every methode of interface is public, so while implementing interface method public keyword is mendatory with interface method
* // inside Q class inherited abstract methode test is there , we have not implemented test inside Q and even we have not made class Q as abstract
* inside interface all the members are public and abstract by default so no need to declare explicitly
* class can implement with any no of interfaces
* class can implement any no super interface is called multiple inheritance
* multiple interface can be implemented using interface

**App24**

* class can extends maximum one super class but implement any no of interfaces
* if class in extending and implementing together then order should be first extends then implements
* one interface can extended with any of interfaces
* only throw interfaces we can achieve multiple interfaces

Method overloading :

* Same method name with different signature is called as method overloading , developing multiple method with same name but different signature(no of args or their return type are different ) for example void test1(), void test1(int a) void test1(int a, int b)
* While overloading a method , method can have any access level, for example , public void test() { }, protected void test(int i) { }
* Class c { void test (int i) { } int test() { return 20;} } while overloading a method return type can be same or different doesn’t matter only signature matters here, same name of the methods but signatures are different
* Abstract class D

{ abstract void test();

Public void test(int i)

{

}

}

* CLASS E { STATIC VOID TEST() { } BOOLEAN TEST (DOUBLE ARG) { RETURN TURE;} } while overloading few can be static and few can be non static
* Interface F { void test(); void test(int i); void test(int I, int j); } inside interface also method can be overloaded
* Interface G { void test(); int test(Boolean b); double test(int i); } is possible with method overloading .
* Class H { void test() { } private int test() { return 20;} } here we will CTE because signature are same return type is different, access level also different but signatures are same
* Refer app24, methodoverloading java file

App25:

* Signature means no of argument should be different doesn’t mean return type, access level , public etc. signature means datatype of the argument or no of the arguments ,
* If no of argument are same and it’s return types are different than also it comes under method overloading
* Inharited method can be overloaded
* in the sub class methode should be implemented not overloaded and if it's not implementing then we have make class as abstract.
* if implementing abstract method to the sub class then we have to use the same signature as in the super class
* interface methods are abstract by default so while implementing to the sub class we have to implement the same signature method

Overriding:

* abstract method of super class implementing to the sub class
* concreat method of super class replacing to the sub class is called overriding and for overriding below rules to be followed
  + signature should be same in order to implement or override
  + return type should be same in order to override and implement
  + access level should be same or wider
  + static should be override to static and non static to non static.
* methode overload and implementation required same signature
* while overriding methode should have same return type
* whenever method is overrided to the sub class and method return type is drived one then while overriding we can choose return type any of return type but should be sub class to that return type 🡺 this concept is called co-variant introduced in jdk 1.5
* while overriding, access level should be same or wider for example super class having default then in the sub class it should be protected but if private then it will CTE
* interface methods are public be default so while implementing interface methods in the sub class or during overrding we have to use public keyword
* while overriding we can use super to super method and here not necessay to use super as first statement

**Casting:**

Converting from one data type to another data type, there are two type of casting is available

1. **Primitive casting :** Converting one numeric primitive to another numeric primitive
2. **Derived casting:** Converting one derived data to another derived data type of same inheritance , one class type to another class type.

8 Primitive data types are available

Numeric Primitive data type:

* byte
* short
* int
* long
* float
* double

**Primitive casting :** below is the order as per the range of the number , byte is most wider narrower and double is most wider.

Generally garbage collector will take care of memory usage.

**byte < short < int < long < float < double**

byte min:-128

byte max:127

short min:-32768

short max:32767

Integer min:-2147483648

Integer max:2147483647

Lower range is overwrite by next higher range.

Under primitive two type casting are available

1. **Auto widening:** it’s automatic. as per the requirement compiler only converting narrower to the required wider means byte to short or byte to int or byte to long or int to double etc, no chance of losing the data .

Ex. Byte b = 10;

int i = b; compiler inside: int i = (int) b;

short s = b compiler inside: short s = (short) b;

long l1 = (long) b; // developer also can do this

1. Explicit narrowing: it’s explicit, compiler is not doing automatically because of chance of loosing some data so developer has to take care for narrowing.

Example:

Short s = 128;

// Byte b = s; // CTE: incompatible data type, narrowing is required

Byte b = (byte) s;

Sop(s) = 128

Sop(s) = -128

Short s1 = 100;

// Byte b1 = s1; // CTE

Byte b1 = (byte) s1;

Sop(s1) = 100;

Sop(b1) = 100;

Note: The value you want to keep inside narrower , we have to take care that wider value is there in the range of narrower .

int I = -32769

//short s = I; // CTE

Short s = (short) i;

Sop(i) = -32769

Sop(s) = 32767// here we are getting different result because short range is till -32768 but we are assigning more value to short so short will go to his max limit and make it 32767

Int I = 20000

// short s = I;

Short s = (short) i;

Sop(i) = 20000

Sop(i) = 20000

So if narrowing is required than go for the narrow one which is having that range.

Eclipse :

* IDE, developed by IBM, it’s free to use no need to buy a license, very widely used
* <http://www.eclipse.org/downloads>
* Prerequisites, jdk installation , and path environment variable settings.
* Font size: Windows🡪 preferences 🡪 General🡪 Apperences🡪 color and fonts🡪 Basics🡪 TextFont
* Always required a project
* Eclipse doesn’t have run time environment so project wise independent run time environment provided by eclipse by integrating some run time jar file in the java project
* By default package name is project name, without package name it’s not advisable to develop java programe.
* While saving java program in eclipse , program will be compiled
* To run java program: ctrl + F11
* Try to make gui so that everything all feature should be there in only one window
* Yellow mark are warnings like variable not used

Autowideing

Narrowing:

* Wider can not be assign to the narrower automatically, CTE
* Whenver narrrwer is hepping data loss is happening, (data change) if value is not fitting than moving to different range
* If very much required then only do the narrowing ,
* Don’t do the Narrower than required then we will get different data so before narrowing plz check the target datatype range.

Derived casting

object

Explicit down casting

Auto upcasting

A

B

C

D

* For Drived casting classes should be in one in heritance , converting one class type to another class type , derived casting also two type
  + Auto up casting
  + Explicit down casting

Autoup casting: any sub class automatically can be converted to any super class type

Explicit down casting: any super class can not be assign to the sub class automatically , we have to casting

Super class type can is assign to sub class type but we have to do the downcasting

A a1 = new A();

B b1 = null;

b1 = new B();

B b2 = b1;

A obj = a1;

// all the above are homogeneous assignment , assignment is proper and no convert ion is required

A a1 = new B();

B b1 = new D();

A a2 = b1;

a1 = new C();

b1 = new D();

Object obj = new C();

// above right side all are sub classes so compiler only will do the required auto up casting

A a1 = (A) new B();

B b1 = (B) new D();

C c1 = (C) new D();

Object obj = (Object) b1;

// Explicit up casting is not mandatory

Explicit down casting:

A a1 = new C();

B b1 = a1; // down casting is required , CTE , explicit casting🡺 B b1 = (B) a1; no CTE

C c1 = a1; // down casting is required , CTE, explicit casting 🡺 C c1 = (C) a1; no CTE

C c2 = b1; // down casting is required , CTE, explicit casting 🡺 C c2 = (C) b1; no CTE

If we will not do the explicit down casting that we will get classCastException , it’s run time exception

* Compiler is checking only data type of the reference variable not checking where it is pointing

Polymorphim:

Without up casting we can not use polymorphism , to achive polymorphism two condition should followed

1. Multiple form of definition
2. Overrding
3. // static members are not involving in polymorphism, throw static method we can achive polymorphism, because reference variable will be
4. // replacing with class name in the case of static member
5. // polymorphism is also called dynamic binding,
6. // mutiple form of a same methode
7. // dynamic dispatch
8. // overriding is necessary
9. // overlading: static binding
10. // overriding: late binding , dispatch

* static method can not be overrided to the sub classes

Final: Final member can be used any no time but can initialize only once , i is declared as final so can be used any no time but can not be reinitialize

Final int i = 10;

Sop(i)

Int j = I;

Sop(i + j);

------------------------------------------------------

Final int I = 10;

I = 10 // CTE, attempting to modify

Sop(i)

===========================================

Final int i

I = 10; // we can initialize the value to final variable but only one not again and again

Sop(i)

==========================================

Final int i ;

I = 0;

I= 0; // CTE

================================================

Final int[] x = new int[10];

X[0] = 20;// ok , X final but x[0] is not final;

Sop(x[0]);

=========================================

Final int[] x = new int[10];

X = new int[10];// attempting to modify x , CTE

Sop(x[0]);

========================================

Class A

{

Final int I;

}

Main {

Final A a1 = new A();

a1 = new A();

}

No CTE, because only int I is the final not a1 is not final

=================================================

Class A {

Final int I ; // final glabal variable can be assign to any value & even it can be 0 as well, even if we will not give any value to final gloabal variable then CTE because there is not default value for the final variable

}

==================================================

**class** B

{

**final** **int** i;

B()

{

i = 10;

}

}

// compile vary successfully

=============================================

**class** C

{

**static** **final** **int** ***i*** = 10;

**static** **void** test()

{

***i*** = 10; // CTE

}

}

============================================

**class** D

{

**static** **final** **int** ***i***;// CTE, gloable final should be initialize while loading to the memory

}

===========================================

**class** E

{

**static** **final** **int** ***i***;

**static**

{

***i*** = 0;

}

}

// compiling successfully

=============================================

**class** A

{

**void** test1()

{

//-----

// -----

}

**final** **void** test2()

{

//===

}

}

// CTE

//method also can be declared as final,

// should not attempt to modify final method in the sub clas or in the same class

//

============================================

**abstract** **class** A

{

**abstract** **final** **void** test();

// final method can not be modify in the sub class

// abstarct means , should be implement in the sub class

// final can not be modify in the sub class

// so method can not be abstract and final together

}

=========================================

// class also can declared as final

// if any claas is a final then we can not devlop any sub class to final class

// cTE

**final** **class** A

{

//=====

}

**class** B **extends** A

{

//------

}  
=====================================================

**abstract** **final** **class** A

{

//=====

// declaring abstract final for a class in not allowed because

// final class should have sub class but abstrac should have sub classs

// CTE

}

=================================================

Eclipse app 11:

Final static: if any variable is declared as final static then it’s perfect constant

Static is loading to the memory one time, inside interface all the attributes are static and final

* any variable static final then identify in a upper case
* any variable static final then identify in a upper case
* any order is ok for public static final keywords but before varible data type and before methode return type else order can be anything
* // by default the attribut of interface are public static and final so we have to initialize explicitly
* // every interface attribute we have to initialize because all the attribute are public static final,
* // perfect constant ==> interface attribute
* // final global variable can not be survive with default value , we have to explicitly assign the any value
* // static members are loading to the memory only one time, if static is a final then only one time
* /\*
* attempting to reinitialize, static final can not be reinitialize
* final keyword can be used for method also, if final methode can not used or overrided to the sub class
* method should not be abstract and final because abstract should be implement to sub class but for final it's not possible
* constructor can not be abstract because constructor can not be inheriting to the sub class
* all constructors are final by default because it's not inheriting to the sub class
* declaring constructors as final will give CTE
* same method can not be abstract and static because static method is the member of the current class and it can not be
* a member of the sub class by implementing , static implementation is only one time not more than one time.
* even though static members are inheriting to the sub class still part of the super class, by default static membre are final,
* we can not modify to develop and will by implementing at the place declaring
* static method we can declare as final but by default it's final only, but we can override to the sub class
* static members are only one time develop
* if class is a final then we can not develop a sub class to that final class
* abstract and final both are illegal combination

**Java Part 2:**

**Eclipse app27**

**Wrapper class:** for any primitive data type there is a wrapper class

Primitive data type Wrapper classes

-------------------- -----------------------------

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Character

boolean Boolean

Why:

* if system or container or method is required only object then we can covert each primitive to object by using it corresponding wrapper class
* if we want to convert primitive to object

Other points:

* all primitives are final
* we cannot develop sub class to the wrapper classes
* every wrapper class is implementing comparable interface based on this wrapper objects based on the contents
* every wrapper class is implementing serializable interface because of this wrapper objects are applicable for seriablization
* in every wrapper class toString() method got overridded because of this we can print wrapper objects content throw SOP
* in all wrapper objects hashcode() and equals() method got overridded because of this we can evaluate wrapper objects based on the content
* By using wrapper classes we can wrap primitive content into object that called boxing
* By using wrapper classes we can un-wrap primitive content from object that called unboxing
* We can use wrapper class to convert primitive into string
* We can use wrapper class to convert String into primitive
* In all wrapper classes there are two constructors available except char primitive data type
* Inside Character class there is only one constructor
* Inside all the wrapper class there are two overloaded value of method except char wrapper class (valueOf)
  + One is taking int type args and other one String type
    - valueOf(String s1)
    - valueOf(int i)
  + inside char wrapper class there is only one valueOf method is available
    - taking corresponding primitive type argument
    - string cannot be converted to char because string having multiple characters while string can be converted to float and double
* We are not developing any task only we will use existing ones

Integer Wrapper class:

public class Integer

{

public Integer(int i)

{

}

public Integer(String i)

{

}

}

* First constructor is taking int type args and other one takes String args
* As a string (.) dot as a special character is allowed for other special char we will get number format issue
* Except char in all the wrapper classes there are two constructor available
* if something is other than true then it will consider as false
* any primitive can be converted to String
* String can be converted to primitive type using parseInt methode of corresponding primitive classs
* any primitive can be converted to String by using wrapper classes because in every primitive wrapper class there is one toString method even char also having toString method
* int i = obj; // right side reference type(class type) and i is primitive type , this kind of assigment was not possible from jdk1.5

// from jdk1.5 and onwards it's allowed ==> compiler is coverting the statement like int i = obj.intValue(); so compiler itself unboxing

* Integer obj = i; 🡺 Error in jdk1.4, primitive to Integer
* In jdk1.5 onwards straight away we can assign integer to class type or class type to primitive because compiler only is doing boxing and unboxing for us.

Object

Number

Double

Float

Integer

Long

short

Byte

With respect to objects there is no relation between primitive types, byte can not coverted to short but Byte object can be converted to Number and number to object.

All above 6 primitive type are belongs or extending from Number class and number extended from Object and can be auto upcasted to Number class. Only in case of primitive auto widening is allowed

* Variable arguments: We can pass any no of arguments including 0, static void test(int ... arg) // var arg, variable argument methodIdentifier(datatype ... arg)
* var arg also introduced in jdk1.5
* one method is var arg and another one is array with same name so duplicate method , we can not overload same method with same signature
* if any var arg is there in a method then it should always be a last argument else CTE
* why last because in if it will be in the middle then not be able to find out how many no of arguments to var args
* same thing with Interface, first extends then implements so we can implement any no of method
* first preference goes to wideing then to boxing
* Second preference goes to auto boxing if no suitable method for auto widing
* Integer auto converted to Number , auto upcasting on the wrapple object level
* last preference goes to var-arg

Generics :

* Introduced in jdk1.5
* For each object , i data type is not changing, so we can say data type is hardcoded
* till jdk1.4 , for each object data type changing was not possible
* class B having 3 attributes , for each one having dfferent data type, so for each object data type will always remain same
* for each object data type is fixed
* If we have the requirement to change the data type for each object then we need Generics
* for c1 obj data type is Integer for c2 obj data type is String
* X identifier , it can be used for data type purpose
* while creating an object we use inside <>
* X identifier , it can be used for data type purpose
* while creating an object we use inside <>
* no arg constructor is not available only argumeneted is available
* no arg constructor is not available only argumeneted is available
* Generics are only for derived data type not for primitive data type , for promitiv data type generics can not be applied
* Generics are applied only for non static members and can not be applied for static members because generics are object base
* and static will not load and use with objects
* \* no arg constructor is not available only argumeneted is available
* \* Generics are only for derived data type not for primitive data type , for promitiv data type generics can not be applied
* \* Generics are applied only for non static members and can not be applied for static members because generics are object base and static will not load and use with objects
* \* Generics representing drived data type only to non static member
* \* Global members always use wrapper class , don't use primitive data type because wrapper (drived data type ) default value is null
* \* drow back with primitive is having some default values
* \* if some thing we want to change from one object to another object then we can use Generics

Inner classes

Enums:

* enum can become a part of jave file
* enum will also generate separate file
* enum are mainly used for representing fixed no of similar kind of ordered constants
* Every constant of the enum should be unique , duplicate constant should not allowed
* Every enum constant should be valid java idendifier
* all enum constant should be separated with comma(, )
* every enum constant having a unique index starting with 0
* inside enum not only enum constant but we can develop any no of attributes, constractors and methods
* each constant of enum is public static final and perfect constant
* There is one super class called Enum and from which some many sub classes are inheirted
* inside enum we can develop attribute, constructor, and methods
* for enum we can not create an object
* inside enum each constant treated as instant or object but not 100% object
* incase of enum whenever enum constant loading to the memory automatically constructor is calling
* Enum also have one minimum one constructor is default one
* why 3 time constructor is called, every enum member is static by default so all static will be loading once class will be loading so enum will be load,
* here each constant treated as one object so when constant loading to the memory for each constant corresponding constructor will be called
* every enum have minimum one constructor
* USE CASE COMMON PROPERTY VARIING FROM ONE CONSTANT TO ANOTHE CONSTANT
* Enum is just only for constants

Class Path:

Dev1

src

Pack1

A.java

D:\Programming\Java\MorningBatch\classpath\_dev3\src>javac -d ../classes –cp D:\Programming\Java\MorningBatch\classpath\_dev1\classes;D:\Programming\Java\MorningBatch\classpath\_dev2\classes C.java

D:\Programming\Java\MorningBatch\classpath\_dev3\src>java -cp ../classes;D:\Programming\Java\MorningBatch\classpath\_dev1\classes;D:\Programming\Java\Mo

rningBatch\classpath\_dev2\classes C

D:\Programming\Java\MorningBatch\classpath\_dev3\src>set classpath=D:\Programming\Java\MorningBatch\classpath\_dev2\classes;D:\Programming\Java\MorningB

atch\c

lasspath\_dev1\clfffasses;../classes

Dev2

src

Pack2

b.java

Dev3

src

C.java

How to create jar file:

* jar cf ../lib/test1.jar \*.java
* javap command parsing a class file and showing the members of the class file
* javac -d ../classes pack1/\*.java pack2/\*.java

Anotation

* Anonotation introduced in jdk1.5 and is to verify our comment, description by the compiler
* Anotations are mainly used for providing a description to package or class or interface or enum or anonotation or fields or methods or constructors , arguments, local variable.
* Annotation can become a member of the java file like classs, interface, enum
* For annotation also class file is generating while compiling
* By default annotation is 100% abstract like interface
* Inside Annotation only abstract methods are allowed
* Annotation is not used for printing anything or executing any logic because it’s only for description purpose , no printing , no execution and no any kind of automation
* Annotation developed with @interface, Annotation is not any kind of keyword we have to use @interface
* Only abstract methods are allowed inside annotations , no iib, sib, constructors , method definition
* Method return type should not have void because we want to show some description using annotation but void will not return anything so we can not show the description that wy can be void . Allowable return type are all primitive, all wrappers(for each primitive there is a wrapper classs), string, Classs (in java.lang package there is one class with name as Class itself so here it can be used as return type purpose ), and we can not go for enum, any drived data type.
* Annotation is allowed only 38 data type as return type , one string , one classs, 8 primitive , 8 wrappers , arrays
* configuration description also can be provided using annotation with run time policy
* Retention is also one of the meta annotation , there are 3 differentional policies are available
  + class // annotation description will go to class file as well because while compiling some binary code will be generating for this policy
  + runtime // annotation description not only moving to the class but moving to the runtime also
  + Source // limited to source file , not be will transferring to the class file by javac command
* For most framework we use annotations with runtime policy
* Mostly we should use annotation outside of source code java file.
* @Inherited is another meta annotation
* @Documented is another meta annotation
* Built in @Override annotation , @Override annotation is not allowed for the method which is not override and we will get CTE
* if we want to override then better use eclipse ctrl+space feature and you can see the auto annotation
* What is Deprecation, class is same but found new logic then old method we can make @Deprecated
* whenever identifying the best way for the existing code then we can deprecate the old method and make a new methods or existing code so don't remove that code but better deprecate that code because running code is already proved and was running and also use @Documented annotation can be use just to show for replacement .
* @SuppressWarnings({"deprecation"}) is used to remove the deprecation warnings
* @SuppressWarnings("unchecked") // don't provide any unchecked wanings inside main method, any kind of warning can be suppressed
* Three built in warnings

Recursive Algo:

* Return will return the execution of it’s caller
* In case of recursive algo, inside one block all the pending statements always executes
* Using recursion we lost the security inspite we can use loops in method and it’s secure , recursion are advisable

DS

b1

b2

b3

d1

d2

D3

e2

e2

String

* package com.lara;
* //Multiple characters can be handled in 2 ways
* //1) Character
* //manipulation is difficult in Characters
* //2) String class, String Buffer, String Builder
* //String is the final class
* //Available in java.lang package
* //We can't develop a subclass to String class as its already a final
* //implementing serializable interface-bcz of this we can serialize String object
* //String class is implementing comparable interface-bcz of this String obj can be sorted based on the content
* //Inside a String class toString method of the Object class got overrided-bcz of this we can print content of the String Object by using its ref var
* //Inside a String class equals and hashcode methods of Object class got overrided-bcz of this
* //we can evaluate 2 String objects based on the content
* //We can create a String Object without a new operator
* //We can add anything to the String through + operator()in C++ there is a operator overloading
* //-additional task provided to same operator)
* //In java no operator overloading
* //Already + operator overloaded with String(Only for Strings)
* //String objects are immutable(we can't change once we create a String Object)
* //Whenever we are attempting to change- with changed content a new object will be generating
* Inside a String class equal and hash code methods of object class got over ridded because of this we can again evaluate string object based on the content.
* We can create a string object without a new operator , we can add anything to the string object using + operator

Note: operator overloading is not available in java but already + is overloaded from java sun developers only for Strings

* String objects are immutable , we cannot change once we create an String objects , whenever we are attempting to change , with the changed content a new object will be generating
* // We can create an String object without new opeator
* // We can print String object content using it's ref variable '
* We can not change String object once created, so here s1 in the line 9 is completly a new object
* We can not change String object once created, so here s1 in the line 9 is completely a new object Because String object is immutable, and first string object s1 in line 7 become abounded
* String class is having one method is there is called length, in case of Arrary length attribute or property is there
* inside string object contains all the character of the string, string object is having indexes of the strings
* We should not call any method using null reference variable and in this case we will get null pointer exception
* String s1 = NULL; // not a keyword so CTE
* String s1 = "null"; // correct one, we will get output as 4
* String s1 = "";// output as 0
* every char has a unique index we can read using charAt
* If index is not available then we will get String index out of range exception
* int index = s1.indexOf('o'); // will print the occurance of first o
* int k = s1.indexOf('z'); // z is not available then we will get -1
* if any char is not available then also we will get -1
* int i = s1.lastIndexOf("made"); // Searching is heppening from right to lefe and search will stop once will first occurance from the first
* int i = s1.lastIndexOf('e'); // single char also can be searched , searching order is changing from rigth to left but indexes are same
* int i = s1.indexOf('e', 9); // Searching is starting from index 10 and in output it will print the index value
* String s2 = s1.substring(0, i); get substring first argument is inclusive(consider) and second argument is exclusive(don't consider)
* String s3 = s1.substring(i + 1); // if not passing second arg then it will take complete rest of the content
* String s3 = s1.substring(i + 1, 15); // 15 index not available then string index out of range excep
* String[] splits = s1.split(";"); // using common separator we can split string into multiple strings
* s1 = s1.toUpperCase(); // while changing any thing a new String object is generating so always store in different object
* System.out.println(s1.length()); // spaces will be counted in length function
* String s2 = s1.trim(); // new object is creating so store it in different object, trim function will remove all the leading and trailing spaces
* System.out.println(s1.startsWith("hello"));
* System.out.println(s1.endsWith("java"));
* System.out.println(s1.equals(s2));
* String object without a new operator, equals will always check the content not matter where the objects are stored
* using new operator string object is creating inside non constant port so there is difference between while creating an object using
* new operator and using string object
* String.equals will always check the content but == here checking object ref block,
* String s2 = s1 + "xyz";, + operator used to concatenation for the strings
* String s4 = s1 + s2; // one ref then new object , == operator will always check the memory location
* s1.concat("xyz"); will create a new object so we have to store in some other object like s1 only we can take

/\* formatters formatting string format is

=====================================================================

%[arg\_index$][flag][width][.Precision] conversion char type

flag

-------

- left alignment

+ explicit +ve sign to the +ve numbers

0 padding with 0 for all left

un occupied places

( to enclose -ve number in the brackets

, to grouping digits in a number

conversion char types

-------------------------------------------------------------------

byte, short, int, long ==> d

float, doublt ===> f

boolean ===> b

char ===> c

String ===> s

======================================================================================================

**Exception Handling**

* java.lang.StackOverflowError: Verifying if all the fields are available or not which was there during the compilation
* **java.lang.NoSuchMethodError**
* **java.lang.NoClassDefFoundError**
* **using eclipse when we compile then complete WS will be compiling so if we want to compile only one file then we have to for command prompt wise compilation**
* Whenever we have any abnormal conditions then either exception or error object will be generating so we should have some code to handle this situation else code flow will be terminating
* To handle these abnormal conditions or in order to continue the codeflow we have some concept called Exception Handling

possibles combinations

* try-catch
* try-catches
* try-catch-finally
* try-catches-finally
* try-finally

Not possible

* try-finally-catch
* try-catch-finally-catch
* Rule : if return is there in finally then after that no return statement is allowed, you will get CTE

Throwable

Error

Exception

Throwable is the super class

* Exception: While running a java prog if any java statement is producing a abnormal condition for that abnormal condition exception object is creating

Error: If any abnormal condition occurred due to external resources not due to java statements then it’s called error like statckmemory, heapmemory, field is not available etc then error object will be generating , error object also can be handling

RuntimeException

Exception

* Every sub class of Error type and run type exceptions are unchecked Exception
* Remaining above are checked exceptions

Unchecked:

* compiler not looking for try and cached and throws while compiling
* no matter try and catch is there or not, compiler will be success, compiler not knowing that there is any exception while running ,
* compiler does have more knowledge about unchecked that why compiler not forcing us to keep try and catch block
* incase of unchecked, compiler have less knowledge
* for example ,null pointer exception , compiler not knowing while compiling but while running compiler will give alert

checked:

* Compiler having more maturity for checked exceptions
* Thread interrupted exception
* String parse exception
* Thorows is mainly for checked exp

File handling:

* Bufferdwriter is implemented on top of filereader so new line

Serialization

* Writing object state into any persisting media can be file system or db is called Serialization and
* If super is implementing serializable then all it’s sub classes eligible for serialization
* With serialization we get decopy feature

Threading

Two wat

Runnable

A

A a1 = new A()

F

Thread

Thread t1 = new Thread(f1)

Object class containing below 3 methods

* wait()
* notify()
* notifyAll()

every class is the sub class of the object class, all class will have all the above 3 methods, so every object will have all the above 3 methods.

Thread can call all the above methods on any object. Whenever thread is calling wait or notify or notifyall , it should have a object lock, this is the condition. And all these methods are non synchronize methods. Before calling any one methods thread should have object lock.

So if thread t1 and t2 is in waiting from the object s1 then they will be under waiting till notify. So if t3 has to call notifyall using object s1 then only t1 and t2 will come out from waiting and else will not release from the waiting list. And t3 has to execute notifyall only notify will not work and also t3 has to use object s1 only not s2.

Without object lock we will get run time error.

Thread on it own not looking for object lock but while wait, notify or notifyall then it’s looking for object lock.

Wait, notify and notifyall should be call from synchronize methods or synchronize blocks

**Thread Pool:**

* Multiple threads in a container is called thread pool, why we need it ?
* Creating and starting thread is very expensive with respect to system resources, whenever using expensive operation it would be better to reuse again and again. So we should have one safer place where we can keep the thread and starting reuse.