THEORY

1. Class: Class is a factory which creates Objects.
2. NEW keyword:

* it tells class to create an object, once the object is created new keyword will get the address and it will be stored in reference variable.
* It will automatically create a default constructor it is mandatory, when ever we write new keyword.

1. Garbage Collector: it helps us to manage the memory in an efficient way, it regularly keeps on removing unused objects so that overflow of memory is avoided.
2. Static Variable:

* Static variables are created outside method inside class, with a prefix of “static”.
* If the variable is static then it can be accessed anywhere in the program, it is like a global variable.
* It can be accessed with 3 types.
  + 1st method: with class name appended with variable name (A.age), (this is a best way to access the static variables).
  + 2nd method: directly by using variable name (there will be some flaw when the local variable has same name as that of the static variable).
  + 3rd method: by creating object and accessing with reference variable, it is bad practice but compiler is smart enough to convert that object to class common memory and access it.
* All these static variables belong or get stored in class common memory.
* Local variable name and static variable name can be same but if we access with directly variable name inside method where we create local variable with same name as local variable then local variable will be printed not static variable so we prefer always accessing using class name appended with variable name.
* We don’t have to initialize variables mandatorily, if not initialized it will get a default value based on data type of variable declared.

1. Non-static Variable:

* Non-Static variables are created outside method inside class, without a prefix of “static”, With mandatory Object creation.
* We have to create Object mandatory so that all non-static variables can only be accessed with object creation and reference variable.
* We don’t have to initialize variables mandatorily, if not initialized it will get a default value based on data type of variable declared.

1. Heap: In JAVA all the objects which are created are stored in Heap memory.
2. Stack: Stack will help us to maintain programs execution flow.
3. HEAP AND STACK CONCEPT: it helps us to maintain memory in efficient way by creating a program with better execution flow and deleting all unused obj with the concept of garbage collector.



1. LOCAL VARIABLE:

* Local Variables are created inside method and can only be used within created method, if they are used outside method, it will give error.
* It should be INITIALIZED mandatorily or else it will give error.
* Local variable can have the same name as that of the static variable.
* Accessing local variable is directly by using its variable name.
* Note: instead of writing “,” to read numbers in program like salary amount we can use “\_” so that it will not give an error.

1. DATA TYPES:

* 9 Types:
  + Byte-> 1 byte -> 0.
  + Short-> 2 byte -> 0.
  + Int-> 4 byte -> 0.
  + Long-> 8 byte ->0.
  + Float-> 4 byte ->0.0.
  + Double-> 8 byte -> 0.0.
  + String(class)-> N/A -> NULL.
  + Char-> N/A -> empty space.
  + Boolean -> N/A -> false.
* 1 Special Type:
  + Var: It is a not a data type because no data type can have a var name as that of the data type but in var we can pass variable name as var also so it’s a type rather than data-type.
  + Var var = 10; (accepted)
  + It was introduced in version 10 and above.
  + It cannot be passed as a method argument
  + It cannot be static or non-static variable it can only be local variable.

1. Reference Variable:

* DATA-TYPE of reference variable is Class name.
* Reference variable can store object address and null values.
* If reference variables are used in method, then it is called local reference variable.
* If reference variables are used in used outside method with static keyword, then it is called global reference variable.

1. Methods:

* A method is **a block of code which only runs when it is** called. You can pass data, known as parameters, into a method. Methods are used to perform certain actions, and they are also known as functions.
* If we call a method with parameters same way, we have to create a method with same argument and data-type should also match
* Special case -> p v test (int… i) {sop(i[index]);}
* A method can have same name as that of the class name.\

1. Return-Keyword:

* It is used in void methods.
* We don’t have to write it mandatorily.
* It will pass control back to the method calling statement.
* We cannot write anything after return keyword it should be the last sentence in the method.
* It can be written inside the main method.
* It can be written in static methods.

1. Return-Value – keyword:

* If used in void method it will give an error.
* It is mandatory to write in non-void methods or it will give an error.
* It will pass control and value back to the method calling statement.

1. Constructor:

* Constructor should have same name as that of class name, its mandatory.
* Every time we write a new keyword a default constructor is created and it is mandatory.
* As many objects we create as many times a constructor is called.
* If we put void in front of constructor then it is not a constructor it is a method, to access it we have to create obj reference variable and call it.
* If we create object with argument, it is mandatory to create a constructor with argument of same data type, it is mandatory if not created it will give an error.

1. JRE (java runtime environment):

* It consists of only java runtime environment.
* It is used by client/customers, where they only have to run the program.

1. JDK (java development kit):

* It consists of both java compile and runtime environment.
* It is used by developers, to compile and run the program or code.

1. CONCEPT OF PROGRAM RENDERING USING UNICODE VALUES:



* Unicode->binary->signal->output.

1. Packages:

* packages are nothing but folder, that is to store all data organized way.
* package names cannot be keywords or java or upper-case.
* we cannot write anything above package it should be the first word in program
* packages are folders created in java to store programs in organised manner.
* To access objects or anything from different package we have to import the package with class name (e.g., p1.A-> where p1 is package name and A is class name.)
* Multiple packages can be created like p1.p2.p3-> that is folder inside folder inside folder.
* When we have 2 packages, where one package has classes A, B, where as other package having class A, then resolve the naming conventions in creating objects in class B we write 2 objects having independently with 2nd method package import concept that is package name followed with class name (p1.A & p2.A) where ever there is A in the program in class B.

1. Constructor overloading:
   * here we create more than one constructor in same class.
   * create constructors with different number of arguments or different type of argument.
   * if those 2 different arguments belong to same data type like int and byte etc.
   * Note: in the below program, it will call int because all numbers by default is treated as int.
   * \*\*\* to pass byte value we write as new //Ex3((byte)10);
   * // \*\*\* to pass long value we write as new //Ex3(10L);
   * // in java if we write any number, it will always be treated as int, if we
   * // explicitly want to pass byte value then we do down casting.
   * Note: class name constructor name method name and variable name can be same as shown in the below example. Although it is not encouraged to write programs this way.
2. Oops concept- Inheritance:

* Inheritance is done for only non-static members.
* Here the class we inherit is called as parent class or super class, and where it gets inherited is called as child class.
* There is no multiple inheritance in class level in java language, i.e., to reduce complications of reading program.
* Extends: is a keyword which helps us to inherit
* Note: we cannot inheritance without importing package where other program is present in other class and other package.

1. Oops concept- Polymorphism:

* It can only be applied on methods and not on variables.
* here we develop a feature such that it can take more than one form.
* There are 2 topics in polymorphism i.e.,1) overloading 2) overriding.

1. Overriding:

* inheritance is mandatory to do this Overriding.
* here we inherit a method from parent class and then we modify the logic of inherited method in child class by once again creating a method with a same signature in the child class
* in Overriding methods won’t get copied but it gets replaced by the new method which we create in child class with same method name, as method is taking 2 forms, we tell here it is undergoing polymorphism.
* Overriding annotation “@Override” helps us to check weather overriding is happening or not. if overriding is not happening then we would get an error.
* Example for Overriding is bank account where there are 2 types membership where for example between gold account and platinum account, we have same method but logic is diff i.e., like benefits are different, such as no of cheque books, rate of interest etc., here methods are same but logic is diff, in this situation we use overriding concept.

Note: here the gold account gets inherited to platinum account but to access as we know here it does not copy items in overriding, we should access gold account members by creating gold account object creation. And all the members which are overridden can be accessed by same common way by creating platinum account object (refer program).

1. Overloading:

* creating more than one method with the same name in same class provided they have diff number of argument or diff type of argument is called as method overloading.
* Example, we should think about email logic, where we send email with attachment and email without any attachment, i.e., here 2 methods are of same name but logic and arguments are different this is best example to understand overloading.

1. IIB:

* IIB are executed when objects are created, number of times we create object same number of times IIB will be called.
* IIB are used to initialize all instance variables in one place & that gives us a better readability of the code.
* Here we can initialize both static and non-static variables.
* MM>IIB-1>IIB-2>constructor>method.
* We can create object in IIB, it won’t give error but it will give exception, i.e., halts program abruptly.

1. SIB:

* runs before main method and it doesn't require any invoking statement.
* no Object creation is required or mandatory to call SIB
* Here only static variables can be initialized inside the SIB, but we cannot initialize non static variables inside SIB it will give an error.
* We can create object in SIB.
* SIB-1>SIB-2>MM>IIB>Constructor>Method.

1. Unary Operator:

* pre-increment/pre decrement: (++i/--i) -here we first increase the value by 1 and stored there itself and it will remain same when we see next i.
* post increment/post decrement:(i++/i--) -here we first don't increase the value by 1 here we store same value but when we see next i value will be increased.
* Example 1:

public class Ex1 {

public static void main(String[] args) {

int i =10;

int j = ++i + i++;// 11, 11++;

System.out.println(i);// 11++ -> 12 output.

System.out.println(j); // 11+11 -> 22 output.

System.out.println("-----------------");

int k =10;

int l = --k + k--;//9,9--

System.out.println(k);//9-- -> 8 output.

System.out.println(l);// 9+9 -> 18 output.

}

}

1. This keyword:

* it’s a special reference variable created automatically to store objects address.
* When object creation happens, one address gets stored in reference variable by us, compiler assigns object address to “this” keyword.
* Using this keyword, we can access non static members of the class.
* when there are several objects created in same class then this keyword will select objects based on which is executing at that point of time, and when there are no objects for this keyword to point then it will go back to first object.
* this keyword points to current object running in the program.
* in non-static method we can access non static members even without this keyword.
* limitations:

a) this keyword cannot be used inside static method; hence the below program throws error.

b) using this keyword we can access static variable, as shown in the example below., it gives warning but not error. its bad practise.

1. Access Modifiers Method/variables:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Private | Default | Protected | Public |
| Same class | Yes | Yes | Yes | Yes |
| Sub class same package | No | Yes | Yes | Yes |
| Non-Sub class same package | No | Yes | Yes | Yes |
| Sub class different package | No | No | Yes | Yes |
| Non-Sub class different package | No | No | No | Yes |

* Private: if you make variable /method private it can be accessed in same class.
* Default: If you make the member default, then it can be accessed anywhere in same package and same class.
* Protected: If you make the variable/method protected, then it can be accessed anywhere in same package and different package only through inheritance.
* Public: if we make variable/method public then it can be accessed anywhere in the program.

1. Access modifies for class:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | private | Default | Protected | public |
| Sub class same package | N/A | Yes | N/A | yes |
| Sub class different package | N/A | No | N/A | yes |

* Private class: Not Applicable.
* Default class: if we make class default then it can only be accessed in same package.
* Protected class: Not Applicable.
* Public class: if we make class public then it can be accessed anywhere.

1. Constructor access specifiers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | private | Default (#) | Protected (#) | public |
| Same class | Yes | Yes | Yes | Yes |
| Diff class same package. | No | Yes | Yes | Yes |
| Diff class diff package. | No | No | No | Yes |

* #: same
* Private constructor: if we make constructor private, then we can create object only in same class, cannot be used outside class or if it is used in different class or package, it will give an error.
* Default constructor: if we make constructor default, then we can create object only in same class and in same package, cannot be used in different package it will give an error.
* Protected constructor: if we make constructor protected, then we can create object only in same class and in same package, cannot be used in different package it will give an error.
* Public constructor: if you make constructor public, then its object can be created anywhere in the program.

1. Constructor chaining:

* This () -> used to call the constructor.
* using this keyword we can call the constructor, but this call should happen only from another constructor as number of constructors, linking keeps on increasing, we call it as constructor chaining.
* Note: we cannot call constructor with this keyword from method.
* while calling a constructor this keyword should always be first statement in another constructor.

1. Getters and setters:

* Are used to initialize variables, without initializing manually with assigning a value or not by assigning value using IIB’s, we initialize with methods these are called as getters and setters, naming convention (private int age-> setAge, getAge).

1. Encapsulation:

* Bundling of data with the method which operates on it, restricting direct access to that variable is called as encapsulation. to restrict direct access of the variable we make variable private to operate on those variables, we create Publicly defined getters and setters.
* data hiding is the part of encapsulation.

1. Type casting: converting particular data type into required data type is called as type casting.

Note: type casting this concept is applicable on static, n/static and local variables.

* Auto up-casting.
  + Here we convert smaller data type to bigger data type.
  + during auto up-casting data loss should not happen.
* Explicit down-casting.
  + here we convert bigger data type to smaller data type.
  + during down-casting data loss might happen. (Syntax int x =10; long y = (long) int =x;).

1. Class casting:

* Class up-casting: here we store child class address into parent class reference variable.

p.c-1

c.c-1 extends p.c.-1 -> p.c-1 p = new c.c-1(); p=new c.c-2();

c.c-2 extends p.c-1

* Class down-casting: here we store parent class object address into child class reference variable.

1. Object: object is the super most class by default in java.

Note: garbage collection method belongs to object.

1. Super Keyword:

* using super keyword, we can access the members of parent class.
* using super keyword, we can access static and non-static members both
* using super () we can call no argument constructor of parent class, given call should be only from child class constructor & super() should be the first argument in the child class constructor, if we don’t write compiler will automatically create by default in every child class constructor to call parent class constructor.
* If we write new keyword in child class, we know that it calls default constructor mandatorily and we also know that every constructor in child class has super() as first statement by default, and this will call parent class no argument constructor, even if we don’t create constructor in child class, new keyword is sufficient enough.
* To call constructor in parent class with arguments, then explicitly we have to write super(with arguments of same data type) in child class constructor with same number of arguments and same data type or else it wont call and it will give error.
* super keyword cannot be used inside static methods.
* we cannot write both super and this in the same constructor as either of statement becomes second statement so it will give error.
* if your child class constructor consists of this keyword then that constructor then super keyword will not be automatically placed.

1. Run time polymorphism: if we do over-riding and class up-casting as shown in the below example (overriding + class up-casting).

* During overriding access specifier scope can be increased but it cannot be decreased as shown in below example.
  + Public -> protected 🡪 not accepted
  + Protected -> public 🡪accepted
  + Etc.
* Note: static members are never inherited, so overriding of that is also not allowed.

1. Abstraction:

* Interface
* ------------

1. Interface:
2. interface can consist of only incomplete methods in it.
3. abstract keyword helps us to define this method is incomplete, not mandatory to write it, we can write it at beginning or after public keyword.
4. it is the blueprint to project.
5. Note: it is like constructing building, i.e. we should have a blue print , same way we should develop a blueprint i.e. which contains all the features which should be developed in project i.e. we incomplete methods which are nothing but Features if we don't use that methods it will give error because class never allows incomplete methods.
6. Note: interface file is also saved as .java only.
7. Note: java is not complete object oriented because we can run programs even without creating objects which is not mandatory.
8. Note: inheritance

\* class -> class --> extends.

\* interface -> interface--> extends

\* interface -> class --> implements

\* class -> interface--> nothing not possible

1. // interfaces in java supports multiple inheritance.
2. //interface 1 -> interface 2-> interface 3-> class Ex2\_3 extends interface 3.
3. --------------------------------------------------------------------------------------------------
4. Static abstract methods cannot be created in interface.
5. Every variable created in an interface by default is final and static.
6. Reference variable of interface can be created but object of interface cannot be created it will give an error.
7. marker interface: an empty interface in java is called as marker as shown below.

Note: on a particular class we can perform extends and implements both, but ensure extends is used first then implements.

1. Final keyword:
2. 1) if you make a variable final then we cannot change its value.
3. 2) if you make static/non static variable as final and if not initialize

then it will give error.

1. 3) all final variable should be written in all upper cases.
2. 4) if you make class as final then inheritance is not allowed.
3. 5) if you make a method final then overriding is not possible.
4. JAVA 8 new features:

* Default Keyword:
  + this keyword helps us to create complete methods inside interface.
* Functional Keyword:
  + In this interface only one incomplete method should be present not zero not more than 1.
  + @FunctionalInterface is annotation used to create a functional interface in interfaces before we create interface.
  + In functional interface we can create many complete methods it wont give error but incomplete method should be only exact 1 in number.
* Lambdas:
  + It helps us to develop code with less lines, this is reason why java is also a functional oriented programming.
  + Limitations: code readability is difficult.
  + Note: lambdas expression is applicable only on functional interfaces.
  + We don’t have to write implements while using lambdas.
  + EX without arguments.
  + EX with arguments, single line syntax.
  + Ex with non-void methods.
  + Example:

public interface Ex8 {

public void test1();

}

public class Ex8\_1 /\*no implements we have to write\*/{

public static void main(String[] args) {

Ex8 ex = (/\* no argument in method \*/) -> { //lambdas syntax

System.out.println(100);

System.out.println(200);

};

ex.test1();

}

}

* Stream API:

IMPORTANT POINTS:

1. Note: instead of writing “,” to read numbers in program like salary amount we can use “\_” so that it will not give an error.
2. Note: we can use string with any data type it wont give error but any other if we use it will give error, (int String = 10; [accepted], int int =10; [error]).
3. Note: Special case -> p v test (int… i){sop(i[index]);}
4. Note: A method can have same name as that of the class name.
5. Note: A local variable can have same name as that of the static variable.
6. Note: we can’t write anything after return keyword, it should be the last argument inside the method.
7. Note: If we put void in front of constructor then it is not a constructor it is a method, to access it we have to create obj reference variable and call it.
8. Note: Every time we write a new keyword a default constructor is created and it is mandatory
9. Note: in the below program, it will call int because all numbers by default is treated as int.

* \*\*\* to pass byte value we write as new //Ex3((byte)10);
* // \*\*\* to pass long value we write as new //Ex3(10L);
* // in java if we write any number, it will always be treated as int, if we
* // explicitly want to pass byte value then we do down casting.

1. Note: class name constructor name method name and variable name can be same as shown in the below example. Although it is not encouraged to write programs this way.
2. Note: only one class should have public when they are created in same file, and which is public should be name of class.
3. Note: we cannot inheritance without importing package where other program is present in other class and other package.
4. Note: When object created happens, one address gets stored in reference variable by us, compiler assigns object address to “this” keyword.
5. Note: using this keyword we can access static variable, as shown in the example below., it gives warning but not error. its bad practise.
6. Note: when there are several objects created in same class then this keyword will select objects based on which is executing at that point of time, and when there are no objects for this keyword to point then it will go back to first object.
7. Note: Data-hiding: here we make variable private so that it cannot be accessed outside class.
8. Note: Hiding-implementation of Data/details: Encapsulation concept.
9. Note: non subclasses: no inheritance but object of parent class created in child class.
10. Note: constructor never undergoes inheritance.
11. Note: we can never use private and protected access modifiers on class, only default and public.
12. Note: protected and Default works in same way when it comes to constructor access modifiers.
13. Note: we cannot call constructor with this keyword from method.
14. Note: data hiding is the part of encapsulation.
15. Note: type casting this concept is applicable on static, n/static and local variables.
16. Note: garbage collection method belongs to object.
17. Note: static members are never inherited, so overriding of that is also not allowed.
18. Note: it is like constructing building, i.e. we should have a blue print , same way we should develop a blueprint i.e. which contains all the features which should be developed in project i.e. we incomplete methods which are nothing but Features if we don't use that methods it will give error because class never allows incomplete methods.
19. Note: if you don’t create child class constructor without argument then compiler will automatically place no arguments constructor with super keyword. by default, we know that whenever we write new keyword anywhere default constructor is created and we know that every constructor in child class created will have a super keyword by default so it will call parent class constructor even if we don't create the constructor and call the parent constructor from child class constructor.
20. Note: we cannot write both super and this in the same constructor as either of statement becomes second statement so it will give error. if your child class constructor consists of this keyword then that constructor then super keyword will not be automatically placed.
21. Note: interface file is also saved as .java only.
22. Note: java is not complete object oriented because we can run programs even without creating objects which is not mandatory.

* Note: inheritance
  1. class -> class --> extends.
  2. interface -> interface--> extends
  3. interface -> class --> implements
  4. class -> interface--> nothing not possible

1. Note: on a particular class we can perform extends and implements both, but ensure extends is used first then implements.
2. Note: lambdas expression is applicable only on functional interfaces.