|  |  |  |  |
| --- | --- | --- | --- |
| Encapsulation | Hiding the data from the Un-Authorized User | E.g. Capsule Tablet Medicine is kept inside the  Capsule to avoid fromexternal factors | POJO  private variable  public getters & setters |
| Abstraction | Highlighting the important properties | E.g. Project Report-Abstract Entire project overview has  been covered in a single page | abstract? |
| Inheritance | Inheriting the parent properties / behavior | E.g. Children’s are inheriting the Parent Assets |  |
| Polymorphism | Many Forms  Same name can be used for different operations  Overloading :  - Same Name / Different Signature | E.g. Command: “ move ”   * + Human: Walk   + Frog: Jump   + Fish: Swim   Command is same for all, but the operations are different. | Overloading:   * add() * add(String) * add(String, String) * add(String, int) |
|  | Overriding:   * Same Name / Same Signature * But different class | Overriding: //Inheritance   * add() :: Class A * add() :: Class B |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Core Java

|  |  |  |
| --- | --- | --- |
| Class | Group of similar properties / operations | E.g. “10th Class” --general english term Properties:   * + students   + teachers Operations: //methods * studying() * teaching() * learning() |
| Object | One Instance of the Class | We get one instance of the class  TenthClass{   * students = “Seetha, Prakathi” * teachers = “Nadhakumar”   o learning()  } |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

|  |
| --- |
| Class – Object |
|  |
| OOPS |
| **Object Oriented Paradigms :**EAIP |

|  |  |  |
| --- | --- | --- |
| Dynamic Memory | * Created at Runtime * Class Variables * [Heap Memory] | ClassA{   * int a * int b * psvm(){   ---- int x;  ----int y;  }  } |
|  | - Created at Compile time |
| Static Memory | * Local variable * [Stack Memory] | a,b: (class -variable or member -variable)  new ClassA(); // memory will be allocated during object creation |
|  |  | x,y: (local -variable )  will be created during compile time itself |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
| Dynamic / Static Memory | | | | |
| - | | | | |
| Java Bean / POJO | | | | |
| JavaBean{  - private membervariables //class variable  o public getters() & setters() //to access private variable  } | | | | |
| Constructor | | | | |
|  | When you create a new instance of | * Default Constructor   new ClassA();   * Parameterized Constructor   new ClassA(5,6); | ClassA{   * int a * int b * ClassA(){   ------ a =0; //default value  -------b =0;  }   * ClassA(int a, intb){   ------ this.a =a; //custom value  -------this.b =b;  }  } |  |
| the class, |
| Constructor will be |
| calledautomatically. |
| This can be used to initialize values in |
| the class |
| Restrictions: |
| - Same Name as Class Name |
| - Only one Default Constructor |
| - Constructor Overloading is |
| possible |
| - When we use Parameterized |
| Constructor, Default Constructor |
| is mandatory |

|  |  |  |
| --- | --- | --- |
| Abstract Class | | |
|  |  | abstract class MathClass{ |
|  | o add(){ |
| When the Child extend Abstract Class,  Compulsory, It has to override the abstract methods. | ----------  }  o abstract subtract(); //abstract method |
| - Planned Hierarchy | //only specifications |
| * More Generalized * abstractMethod(); [Only Specification] | } |
| - abstract methods can be available only at abstract classes | ClassA extends MathClass { |
| * Abstract Class can have   + (abstract methods and normal methods)   + (Specifications & Implementations) | o subtract(){  ------MUST OVERIDE------  -----abstractMethods()---- |
|  | } |
|  | } |
|  | new ClassA(); // is possible |
|  | new MathClass(); //is not possible |
|  | // is possible |
| Cannot create object for abstract class. If we create override while creation itself.  Childs Objects can be created | new JAbstarctClass1() {  subtract () {  ------MUST OVERIDE------  -----abstractMethods()---- |
|  | } |
|  | }; |
|  | | |
| Interface | | |
|  | Interface = 100 % Abstract Class | Interface MathInterface{   * add();//only Specifications * subtract(); //by default all are abstract methods   public static final String PI =”3.14”; //constants  }  ClassA implementsMathInterface {   * subtract(){   ------MUST OVERIDE------  -----abstractMethods()----  }  }  new ClassA(); // is possible  new MathClass(); //is impossible |
| - Only Specifications //Only abstract method definitions |
| - No Implementations //No Normal methods |
| - In Interface, |
| o By default all are public [abstract] methods |
| o It can have public static final variables |
| //constants |
| - |
| - Not Possible to create object for Interface. |
| - Not possible to have static or finalmethods in Interface |
| Why Interface? |
| - Planned Hierarchy |
| - Gives Structure to the Class |
| E.g. Skeleton |
| - Using Human Skeleton, we can make only Humans |
| - Using Fish Skeleton, we can make only Fishes |
| - Using Frog Skeleton, we can make only Frogs |
| So Interfaces provides strict structure toimplement classes |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inheritance Ways | | | | |
| * Class(can extend ) Class * Interface (can extendmultiple) Interface * Class  (can implement multiple) Interface * Abstract Class  (can implement multiple)  Interface //It doesn’t require to override all the Interface methods | | | | |
|  |  |  | |  |
|  | Why Multiple Inheritance is not possible in Java?   * Due to “Diamond Problem” * One class cannot extend more than one class   How to implement Multiple Inheritance indirectly in Java?   * Using ‘Interface’ * One Class can implement multiple Interfaces * E.g. Hybrid * Human Skeleton + Fish Skeleton = [Fish & Human] Hybrid | |  | |

|  |  |  |
| --- | --- | --- |
|  | C:\Users\jpalania\Desktop\d1.png | |
| final -keyword | | |
| * final methods cannot be overridden * final variables cannot be modified * final Class cannot be inherited / sub classed   + Many Java Standard Library Classes are final   + E.g. java.lang.System , java.lang.String | | |
| Polymorphism | | |
| Many Forms  Same name can be used for different operations 2 Types:   * Static / Compile Time / Early Binding // Method Overloading * Dynamic/ Run Time / Late Binding //Method Overriding | | |
|  | Method Overloading:   * Same Name * Different Signature * Differentiated by   + No. of arguments   + Type of the arguments | Class A{   * add(){   }   * add(String) (){   }   * add(String, String) (){   }   * add(int, int) (){   ----------  }  }  Class B {   * psvm(){   ClassA a = new ClassA(); a.add();  a.add(“hi”) |

|  |  |  |
| --- | --- | --- |
|  |  | a.add(5,8);  }  } |
|  |  | Class A{ |
|  | o add(int, int) (){ |
|  | class A add() |
| Method Overriding: | }  } |
| * Same Name * Same Signature * In Inheritance * Decision taken at Runtime | Class B extends ClassA{  o add(int, int) (){  -------class B---add()----  } |
|  | } |
| o Variable also can be overridden, called Shadowing or Hiding. | Class C { |
| o Private methods cannot be | o psvm(){ |
| overridden | ClassA a = new ClassA(); |
| o Using ‘super’ keyword, Child class | ClassB b = new ClassB(); |
| can access parent Class overridden methods | a.add(“hi”,”Shankar”); //-------class A---add()----  b.add(“hi”,”Shankar”); //-------class B---add()---- |
|  | } |
|  | } |
| Packages | | |
| o Folders to organize our files package com.divya; //defining package  import com.divya.JClassA //importing class from different package  importjava.lang.\*; //by default all java programs are imported java lang packages. | | |
| Access Modifiers | | |
|  | | |

|  |  |
| --- | --- |
| private | o Accessed by within the class |
| default | Accessed by   * within the class * anyclass within the package |
| protected | Accessed by   * within the class * any class within thepackage * only child classfrom different package |
| public | Accessed by   * Any Class / Any Package * But within JVM |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| Other Modifiers |
|  |
| Exception |
| Error Exception   * Un expected situation o Expected / Un expected situation * occurs during Program Execution o occurs during Program Execution * further execution -impossible o further execution is-possible   - We can able to handle exception and do necessary action accordingly.   * + By default, JRE will handle the exception   + User can also handle exception manually   - try/catch/throw/finally  **Exception Types:**  Checked Exception Un-checked Exception   * Compile time Exception o Run Time Exception * Will be known at compile time itself o Will be known only at program runtime E.g. * E.g. o E.g. * FileNotFoundException o NullPointerException * ClassNotFoundException o NumberFormatException * IOException o ArrayIndexOutOdBoundException   **Exception Tree:** |

|  |  |
| --- | --- |
| final | o final class / method / variable |
| abstract | o abstractclass / method |
| static | * static method/ variable * static member can access only static members directly * Any Class can access static members without creating object   + By using ClassName   + ClassA.add();   + ClassA.MATH\_CONSTANTS; * A class can have only one copy of static methods and variables |
| native |  |
| transient |  |
| synchronized | * synchronized method * to control access in Multi-Threading Programs |
| volatile |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |

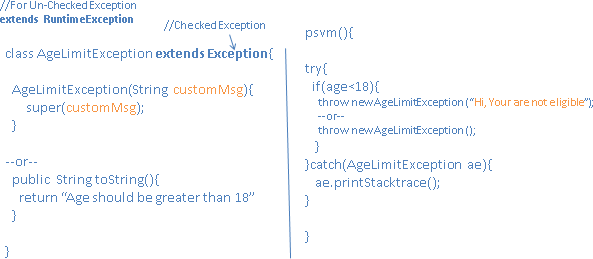
|  |  |
| --- | --- |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
| **Exception Handling Types:**   * try / catch /finally * throws | | |
|  | throws |  |
|  | * If I am not willing to handle the exception, But I am still want to run the program * throws the exception occurred to the invoked method myMethod() throws Exception1,Exception2{ |

|  |  |
| --- | --- |
| try | o here we can keep Exception expected code |
| catch | o here we can handle that Exception depending on Exception type |
| finally | o This block will be called, irrespective of exception occurred or not. |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | }  If Checked Exception:   * If method is throwing checked exception,compulsory the invoked methods must handle that exception   If Un-Checked Exception:   * If method is throwing un-checked exception, the invoked methods need not to handle that exception |  |
| User Defined Exception   * If I am not willing to handle the exception, But I am still want to run the program * throws the exception occurred to the invoked   method | | |
| Multi-Threading | | |
| * A single process can splits into multiple threads * those threads can be run parallel * It is depending on OS * E.g. Need not to wait for the User Input, parallel we can do something which is not relevant to the user input   **How to implement Multi-Threading in Java?**   1. implements Runnable 2. extends Thread   **1.implements Runnable** | | |



|  |
| --- |
|  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| **2.extends Thread** | | |
|  | Thread Life Cycle |  |
|  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | | | |  |
|  |  | | | |
|  | sleep | o setting the [time limit] to wait & execute |  |
| wait | o no time limit [wait until ‘notify’ comes] |
| blocked | o execution is stopped or blocked by the owner |
|  | **Synchronization:**  In Java every object has a ‘LOCK’  Using Synchronization we can enable the LOCK  //TODO  **Daemon Thread:**   * Thread (that worksin)  Background (to support) the current Execution * E.g.   + Garbage Collector   + Clock Handler * By default, threads are NOT a Daemon thread * How to make normal Thread as Daemon Thread?   + setDaemon(true); * How to check the Thread is Daemon or not?   + isDaemon() | | | |

|  |
| --- |
| Collections |
| Collections:   * A group of objects treated as single entity * Collections Framework has,   + Interfaces (7-core-Interfaces)     - Others are Utility Interfaces       * Comparable       * Comparator       * Iterator       * Enumerator   + Implementations () |

|  |  |  |  |
| --- | --- | --- | --- |
| **HashMap Vs TreeMap Vs LinkedHashMap :**  All three classes implement the Map interface Differences:  HashMap makes absolutely no guarantees about the iteration order. It can (and will) even change completely when new elements are added.   * TreeMap will iterate according to the "natural ordering" of the keys according to their compareTo() method (or an externally supplied Comparator). Additionally, it implements the [SortedMap](http://java.sun.com/javase/6/docs/api/java/util/SortedMap.html) interface, which contains methods that depend on this sort order. * LinkedHashMap will iterate in the order in which the entries were put into the map   ["Hashtable"](http://en.wikipedia.org/wiki/Hashtable) is the generic name for hash-based maps. In the context of the Java API, Hashtable is an obsolete class from the days of Java 1.1 before the collections framework existed. It should not be used anymore, because its API is cluttered with obsolete methods that duplicate functionality, and its methods are synchronized (which can decrease performance and is generally useless). Use [ConcurrrentHashMapi](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentHashMap.html)nstead of Hashtable. | | | |
| Other Utilities in -[Collection Framework] | | | |
|  | Comparable | Comparator |  |
|  | * compareTo(Object o) * If we have a source file to modify the content | * compare(Object o1, Object o2) * If we do not have access to source file to modify the content |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **What are the ways to iterate Collections elements?** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Enumerator | * It is not part of Collection Framework, part of java.util.\* package * Collection of value can be iterated * But cannot modify the values * Unidirectional | boolean hasMoreElements()  Object nextElement() | Vector v = new Vector(); Enumeration e = v.elements(); while(e.hasMoreElements){ MyClass m = (MyClass)e.nextElement();  } |
| Iterator | * It is part of Collection Framework * Collection of value can be iterated * Values can be modified * Unidirectional | boolean hasNext() Object next() Object remove() | List l = new ArrayList(); Iterator i =l.iterator(); while(i.hasNext){  MyClass m = (MyClass)e.next();  } |
| ListIterator | * It is part of Collection Framework * Collection of value can be iterated * Values can be modified * Bidirectional | boolean hasNext() Object next() Object previous() Object remove() | List l = new ArrayList(); ListIterator li = l.listiterator(); while(li.hasNext){  MyClass m = (MyClass)e.next();  } |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
|  |
| Why wait() ,notify() and notifyAll() methods are in ‘Object’ class? \*\*\*\*\*\*  -- instead of ‘Thread’ class? |
| * Because of ”Locking” mechanism * Each and every object needs a lock * When we do “Synchronization” in Multi-Threading * The running thread will hold the locksof the common object/resources--until the execution completes * So these locks are applied on Object level –not atThread level   wait() ,notify() and notifyAll(): //releases the lock   * wait()//(releases the lock) (make the thread suspended) * notify()//(releases the lock) (resumesa thread which is suspended) * notifyAll() //(releases the lock) (resumes all thread which are suspended)   Real Example: “Bank-Joint Account” (2 Users) Balance: Rs.2000  Scenario:  First User trying to purchase worth of Rs.1500  At the same time Second User trying to withdraw Rs.1500 from ATM  How it works:   * Here whichever the channel first access the account to perform the transaction will acquires a lock * The Other channel will wait for the lock. * The channel which is waiting for lock status is not aware of which channel acquires the lock * At the same time the channel which is already acquired the lock is not aware of that who are waiting to acquire lock on the particular account. * Here “lock” is applied on the “account” and “not on channel”   class MyThread1 **extends** Thread**{** private Emp e**;**  private Emp e**;** private Emp e**;**  Mythread1**(**Emp e**){ this.**e**=**e**;**  **}**  publicsynchronizedvoid run**(){** synchronized**(**e**){**//Locks 'e' object  //  sysout**(**e**.**getEmpId**());**  //  //locks 'e' object until this synchronized block execution completes  **}**  **}**  **}**  class MyThread2 **extends** Thread**{** private Emp e**;**  Mythread1**(**Emp e**){ this.**e**=**e**;**  **}**  publicvoid run**(){** |

|  |
| --- |
| **this.**sleep**(**"20000"**);**//this waits for 2000 milliseconds + holds the lock of the Emp e object  **}**  **}**  class MainClass**{** psvm**(){**  Emp e **=new** Emp**(**"1001"**);** Thread t1 = new MyThread1(e); Thread t2 = new MyThread2(e);  t1.start();  t2.start();  }  } |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

EAIP

# OOPS

(tutorialspoint)

|  |
| --- |
| OOPS:   * Encapsulation * Abstraction * Inheritance   + SimpleInheritance   + Multi Level Inheritance   + Multiple Inheritance (Java doesn’t support this) * Polymorphism   + Method Overloading   + Method Overriding * Interfaces * Packages |
| Encapsulation |
| * Hiding the data from the Un-Authorized User * Real-timeE.g.“Capsule Tablet”   + Medicine is kept inside the Capsule to avoid from external factors * Java E.g.“POJO classes”   + POJO : Plain Old Java Object   + private variables // public getters and setters to access that variable   **class** Emp{  **privateint**empId;  **publicint** getEmpId() {  **return**empId;  }  **publicvoid** setEmpId(**int** empId) {  **this**.empId = empId;  }  }   * If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. * For this reason, encapsulation is also referred to as“Data Hiding” |
| Abstraction |
| * Highlighting the important properties * Real-time E.g.“Project Report –Abstract”   + Entire project overview has been covered in a single page * Java E.g. “Abstract Class” * abstract class is like normal class   + we cannot create object for abstract class (cannot be instantiated)   + it can have both abstract method and normal method     - abstract method: doesn’t contain body (just definition) |

|  |
| --- |
| **Then What is the use of having abstract class?**   * abstract class is mainly used to define a structure when it comesto inheritance   + whatever the child class extending abstract class   + it must override the abstract method of the class   **Then How do we access the abstract class normal method?** (since we cannot instantiate object)   * Child class inherited the abstract class and overridden all the abstract methods * Now we can able to create object for Child Class. * From this child class object we can access the normal method of abstract class (Parent Class) |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |