**Java: Day2**

**// METHOD CALLING INHERITANCE, POLYMORPHISM. ABSTRACT METHODS.**

[**https://www.eduonix.com/blog/java-programming-2/learn-to-implement-polymorphism-inheritance-abstract-methods-overriding-and-overloading-in-java/**](https://www.eduonix.com/blog/java-programming-2/learn-to-implement-polymorphism-inheritance-abstract-methods-overriding-and-overloading-in-java/)

create classes under multiple packages

calling classes under different packages

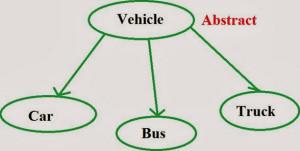
write code to handle exceptions with try/catch/finally

what is final keyword

write code for interface and create class to implement that interface

write code for creating abstract class

**Abstract : –** When a class contains one or more abstract methods, then it should be declared abstract. You must use abstract keyword to make a class abstract. We cannot use abstract classes to instantiate objects directly. It needs to be extended and its method needs to be implemented. The abstract methods of an abstract class must be defined in its subclass. You cannot declare abstract constructors or abstract static methods. Abstract class have both method abstract as well as non-abstract methods. It also has a member variables and constructors.



***Example : This example shows how abstract is used in a java.***

***abstract class A //class declare as abstract  
{ //declare method as abstract  
abstract void callme();  
}  
  
  
class B extends A  
{ //inherit callme method  
void callme()  
{  
System.out.println("This is callme");  
}  
public static void main(String[] args)   
{ //create object of class B  
B b = new B();  
//call method  
b.callme();  
}  
}***

***Output :***

***This is callme***

***Abstract Method : Abstract method does not have any body. It is always ends with (;) semicolon. Abstract method must be overridden. It must be in an abstract class. It can never be static and final. Abstract methods are those which need to be implemented in subclass. If class has one abstract method then whole class is declared as abstract. Private method cannot be abstract.***

***Example : This example shows how abstract method is used in a class.***

***abstract class Calculator  
{ //define 2 integers  
protected int no1;  
protected int no2;  
//declare abstract method  
abstract int sum();  
}   
  
class Addition extends Calculator//extends with Superclass  
{  
Addition (int n1, int n2)//constructor of Addition  
{  
no1 = n1;  
no2 = n2;  
}  
int sum()//define method  
{  
return no1 + no2;//return Addition  
}  
}   
  
class Subtraction extends Calculator//extends with Superclass  
{  
Subtraction (int n1, int n2)//constructor of Subtraction  
{  
no1 = n1;  
no2 = n2;  
}  
int sum()//define method  
{  
return no1 - no2;//return Subtraction  
}  
}  
  
class Multiplication extends Calculator//extends with Superclass  
{  
Multiplication (int n1, int n2)//constructor of Multiplication  
{  
no1 = n1;  
no2 = n2;  
}  
int sum()//define method  
{  
return no1 \* no2;//return Multiplication  
}  
}  
  
class AbstractMethodDemo  
{  
public static void main(String args[])//main method  
{  
Addition a = new Addition (5, 8); //Create object of Addition  
Subtraction s = new Subtraction (32, 16); //Create object of Subtraction  
Multiplication m = new Multiplication (4, 2); //Create object of Multiplication  
  
System.out.println ("Sum of Addition: " + a.sum ()); //Call method Addition  
System.out.println ("Sum of Subtraction: " + s.sum ()); //call method Subtraction  
System.out.println ("Sum of Multiplication: " + m.sum ()); //call method Multiplication  
}  
}***

***Output :***

***Sum of Addition : 13  
Sum of Subtraction : 16  
Sum of Multiplication : 8***

implement method overloading

***Method Overloading : It is nothing but in the same class, if name of the method remains same but the number and type of arguments or parameters are different, then it is called as method overloading. This concept is used for compile-time. Present in the same class. And can have different return types. It helps in maintain consistency in method naming, doing same task with different parameter. It helps to reduce overhead. It is also known as static polymorphism. Static method can be overloaded. Static binding is used for method overloading. It gives better performance than method overriding. Private and final methods can be overloaded. In method overloading return type should be same as the other methods of the same name. In method overloading argument list should be different.***

***Example : This example shows how method overloading is used in a java.***

***package MethodOverload;  
  
class Children //create class  
{  
void student()//define method without parameters  
{  
System.out.println ("Students are playing");  
}  
//define same method with parameter  
void student(int rollno, String name)  
{  
System.out.println ("Roll No: "+rollno+"\nName: "+name);  
}  
public static void main(String[] args)//main method  
{  
Children c = new Children (); //Create object of class  
c.student (); //without parameters method call  
c.student (101, "Priya"); //with parameters method call  
}  
}***

***Output :***

***Students are playing  
Roll No: 101  
Name: Priya***

**implement method overriding**

**Method Overriding :** Method overriding is nothing but the method in the child class should have the same name, same signature and parameters as the one in its parent class and also have the same return type. If a method declared final then it cannot be overridden. If a method declared static then it cannot be overridden but it can be re-declared. If a method cannot be inherited, then it cannot be overriden. It is used for runtime polymorphism. It must be is-a relationship. Polymorphism is applied on method overriding. It is a run-time concept. Abstract methods must be overridden. Constructors cannot be overridden. Dynamic binding is used for method overriding. Private and final method cannot be overridden.

***Example : This example shows how method overriding is used in a java.***

***package MethodOverride;  
  
class Children //parent class  
{  
public void play()//define method  
{  
System.out.println ("Children can play cricket");  
  
}  
  
class Student extends Children//extend parent class  
{  
public void play()//override method  
{  
System.out.println ("Students can play football");  
}  
}  
  
class Test  
{  
public static void main(String[] args)//main method  
{ //create object of Children and Student class  
Children c1 = new Children ();  
Student s = new Student ();  
//call method play  
c1.play ();  
s.play ();  
}  
}***

***Output :***

***Children can play cricket  
Students can play football***

**implementing polymorphism**

***Polymorphism : It is nothing but the ability to take more than one form. This can be applied to both operations as well as objects. It is tightly coupled inheritance. It means one interface, many possible implementations.***



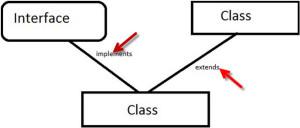
***Two types of Polymorphism:***

1. ***Static Polymorphism: It is achieved through function overloading and operator overloading. It is always faster. It is also called as compile time polymorphism. Example of static polymorphism is method overriding using final or private methods. At the compilation time java knows which method is call by checking the arguments, so it is also known as early binding or static binding.***
3. ***Types of static polymorphism :***
4. * ***Function Overloading: It is nothing but the ability of one function performs different tasks. These functions must differ by the data types. To call function the same function name is used for various instances.***
   * ***Example : This example shows how function overloading is used in a java.***
   * ***package MethodOverload;  
       
     class Children //create class  
     {  
     void student()//declare method without parameters  
     {  
     System.out.println("Students are playing");  
     }  
     //declare same method with parameter  
     void student(int rollno, String name)  
     {  
     System.out.println("Roll No: "+rollno+"\nName: "+name);  
     }  
     public static void main(String[] args)//main method  
     {  
     Children ch = new Children();//Create object of class  
     ch.student();//without parameters method call  
     ch.student(1, "Priya");//with parameters method call  
     ch.student(2, "Pranjal");  
     }  
     }  
     Students are playing  
     Roll No: 1  
     Name: Priya  
     Roll No: 2  
     Name: Pranjal***

   * ***Operator Overloading : Java does not support operator overloading.***
6. ***Dynamic Polymorphism : It is also called as run-time polymorphism. In this case java compiler does not know which method is invoked at compilation time. Just JVM decides which method is invoked at the run-time. Method overriding is example of run-time polymorphism. In this case overridden method is invoking through the super class reference variable.***
8. ***Types of Dynamic Polymorphism in java :***
9. * ***Virtual Function: This is nothing but the function whose performance can be overridden within an inheriting class by a function with the same argument or signature. Virtual function cannot be declared as private. In this function we get warning if we do not use Virtual or New keyword. You can use new keyword rather than Virtual.***
11. ***Example : This example shows how dynamic polymorphism is used in a java.***
13. ***package MethodOverride;  
      
    class Children //parent class  
    {  
    public void speak()//define method  
    {  
    System.out.println("Children speak in Hindi");   
    }  
    }  
      
      
    class Student extends Children//extend parent class  
    {  
    public void speak()//override method  
    {  
    System.out.println("Students can speak in English");  
    }  
    }  
      
      
    class Test  
    {  
    public static void main(String[] args)//main method  
    { //create object of Children and Student class  
    Children c1 = new Children();  
    Student s = new Student();  
    //call method speak  
    c1.speak();  
    s.speak();  
    }  
    }***
14. ***Output :***
16. ***Children speak in Hindi  
    Students can speak in English***

**implementing interface**

**Interface : –** Interface is nothing but the collection of constant values and method definitions without implementations. In this case using an interface you can capture similarities between unrelated classes without forcing relationship between classes. A class can access only from one class but it can implement more than one interface. If we define a new interface you are defining a new reference data type. Functions of interface should be public and abstract. A class implementing an interface must use the keyword **implements**. Object of an interface cannot be created. Field of an interface must be final and public. Interface cannot be instantiated because interface does not have constructors.



***Example : This example shows how an interface is used in a class.***

***class InterfaceDemo //create main class  
{  
//main method  
public static void main(String[] args)   
{  
//create object of class Student  
Children chstud = new Student();  
chstud.play();//call method  
}  
}  
interface Children//create interface  
{   
public void play();//declare method  
}   
class Student implements Children//implements interface  
{  
public void play()//define method  
{  
System.out.println("Students are playing football");  
}  
}***

***Output :***

***Students are playing football***

***write a code to save data into excel file and read from excel file (POI and jexcel API)***

**package excelreader;**

**import java.io.FileInputStream;**

**import java.io.FileNotFoundException;**

**import java.io.IOException;**

**import org.apache.poi.EncryptedDocumentException;**

**import org.apache.poi.openxml4j.exceptions.InvalidFormatException;**

**import org.apache.poi.sl.usermodel.Sheet;**

**import org.apache.poi.ss.usermodel.Workbook;**

**import org.apache.poi.ss.usermodel.WorkbookFactory;**

**public class Excelsheets {**

**/\*\***

**\* @param args**

**\*/**

**static int RowCount=0;**

**public static void main(String[] args) throws EncryptedDocumentException, InvalidFormatException, IOException {**

**// TODO Auto-generated method stub**

**FileInputStream fis=new FileInputStream("//home//rahul//Desktop//sheeet1.xlsx");**

**Workbook wb=WorkbookFactory.create(fis);**

**org.apache.poi.ss.usermodel.Sheet s= wb.getSheet("Sheet1");**

**RowCount =s.getLastRowNum();**

**System.out.println(RowCount);**

**}**

**}**

**package excelreader;**

**import java.io.FileInputStream;**

**import java.io.IOException;**

**import org.apache.poi.EncryptedDocumentException;**

**import org.apache.poi.openxml4j.exceptions.InvalidFormatException;**

**import org.apache.poi.ss.usermodel.Cell;**

**import org.apache.poi.ss.usermodel.Row;**

**import org.apache.poi.ss.usermodel.Workbook;**

**import org.apache.poi.ss.usermodel.WorkbookFactory;**

**public class ReadngfromexcelSheet {**

**/\*\***

**\* @param args**

**\*/**

**static int RowCount=0;**

**public static void main(String[] args) throws EncryptedDocumentException, InvalidFormatException, IOException {**

**// TODO Auto-generated method stub**

**FileInputStream fil=new FileInputStream("//home//rahul//Desktop//sheeet1.xlsx");**

**Workbook wb=WorkbookFactory.create(fil);**

**org.apache.poi.ss.usermodel.Sheet s=wb.getSheet("Sheet1");**

**Row row=s.getRow(2);**

**Cell cl=row.getCell(0);**

**System.out.println(cl);**

**String Cellvalue=cl.getStringCellValue();**

**System.out.print(Cellvalue);**

**}**

**}**

**package excelreader;**

**import java.io.FileInputStream;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import org.apache.poi.EncryptedDocumentException;**

**import org.apache.poi.openxml4j.exceptions.InvalidFormatException;**

**import org.apache.poi.ss.usermodel.Cell;**

**import org.apache.poi.ss.usermodel.Row;**

**import org.apache.poi.ss.usermodel.Sheet;**

**import org.apache.poi.ss.usermodel.Workbook;**

**import org.apache.poi.ss.usermodel.WorkbookFactory;**

**public class writingininexcelfile {**

**/\*\***

**\* @param args**

**\* @throws IOException**

**\* @throws InvalidFormatException**

**\* @throws EncryptedDocumentException**

**\*/**

**public static void main(String[] args) throws EncryptedDocumentException, InvalidFormatException, IOException {**

**// TODO Auto-generated method stub**

**FileInputStream fiu=new FileInputStream("//home//rahul//Desktop//sheeet1.xlsx");**

**Workbook wb=WorkbookFactory.create(fiu);**

**Sheet s=wb.getSheet("Sheet1");**

**Row r=s.getRow(0);**

**Cell c=r.getCell(0);**

**c.setCellType(c.CELL\_TYPE\_NUMERIC);**

**c.setCellValue(140);**

**FileOutputStream fis=new FileOutputStream("//home//rahul//Desktop//sheeet1.xlsx");**

**wb.write(fis);**

**fis.close();**

**System.out.println("sucess");**

**}**

**}**

***write code to add items to integer, string array***

[**http://tutorials.jenkov.com/java/arrays.html**](http://tutorials.jenkov.com/java/arrays.html)

**package** arrayss;

**import** java.util.ArrayList;

**public** **class** AddiigElemntAtendofArray {

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

// **TODO** Auto-generated method stub

// Create an ArrayList that holds references to String

ArrayList<String> names = **new** ArrayList<String>();

// Capacity starts at 10, but size starts at 0

System.***out***.println("initial size: " + names.size() );

// Add three String references

names.add("Amy");

names.add("Bob");

names.add("Cindy");

System.***out***.println("new size: " + names.size() );

// Access and print out the Objects

**for** ( **int** j=0; j<names.size(); j++ )

System.***out***.println("element " + j + ": " + names.get(j) );

}

}

***write code to retrieve items from integer, string array***

***write code to add items to ArrayList collection***

***write code to retrieve items from arraylist (using for each loop***\_

USING FOR EACH LOOP:

**package** arrayss;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**public** **class** UsingForEachloopinArrays {

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

// **TODO** Auto-generated method stub

ArrayList<String> crunchifyList = **new** ArrayList<String>();

// add 4 different values to list

crunchifyList.add("eBay");

crunchifyList.add("Paypal");

crunchifyList.add("Google");

crunchifyList.add("Yahoo");

crunchifyList.forEach((hi)->{

System.***out***.print(hi);

});

}

***LOOPING ARRAY LIST IN JAVA***

package com.mkyong.core;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

public class ArrayListLoopingExample {

public static void main(String[] args) {

List<String> list = new ArrayList<String>();

list.add("Text 1");

list.add("Text 2");

list.add("Text 3");

System.out.println("#1 normal for loop");

for (int i = 0; i < list.size(); i++) {

System.out.println(list.get(i));

}

System.out.println("#2 advance for loop");

for (String temp : list) {

System.out.println(temp);

}

System.out.println("#3 while loop");

int j = 0;

while (list.size() > j) {

System.out.println(list.get(j));

j++;

}

System.out.println("#4 iterator");

Iterator<String> iterator = list.iterator();

while (iterator.hasNext()) {

System.out.println(iterator.next());

}

}

}

***write code to add items HashMap***

<https://www.tutorialspoint.com/java/util/java_util_hashmap.htm>

<http://www.careerbless.com/samplecodes/java/beginners/collections_ds/inertandretrieveMap.php>

**package** arrayss;

**import** java.util.HashMap;

**public** **class** Hashmapadding {

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

// **TODO** Auto-generated method stub

// create hash map

HashMap newmap = **new** HashMap();

// populate hash map

newmap.put(1, "tutorials");

newmap.put(2, "point");

newmap.put(3, "is best");

System.***out***.println("Map value before change: "+ newmap);

// put new values at key 3

String prevvalue=(String)newmap.put(3,"is great");

// check returned previous value

System.***out***.println("Returned previous value: "+ prevvalue);

System.***out***.println("Map value after change: "+ newmap);

}

}

***The Output You can see is:***

Map value before change: {1=tutorials, 2=point, 3=is best}

Returned previous value: is best

Map value after change: {1=tutorials, 2=point, 3=is great}

***write code to retrieve items HashMap***

***Good link for HashMap more methods of writing***

[***http://www.java2novice.com/java-collections-and-util/hashmap/all-keys/***](http://www.java2novice.com/java-collections-and-util/hashmap/all-keys/)

HashMap<Integer,String> First=**new** HashMap<Integer,String>();

First.put(45, "Rahul");

First.put(66, "Hello");

First.put(77, "Kiing");

//First.remove(66);

First.replace(77, "JAi");

//ths is print the whole set of the map

System.***out***.println(First);

//to get particular element

System.out.println(First.get(give key number))

//for Iterating through HashMAP

Set<Integer> k=First.keySet();

//Using the Advanced For loop to retrive each key from the map set

**for**(Integer key:k){

System.***out***.println(key);

}

***Write code to add items to hashset***

**package** arrayss;

**import** java.util.HashSet;

**public** **class** HasssetAddingItems {

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

// **TODO** Auto-generated method stub

HashSet A=**new** HashSet();

A.add(1);

A.add("king");

A.add("me");

A.add(998);

System.***out***.print(A);

}

}

OUT PUT:

[1, king, 998, me]

***Write code to retrieve items to hasset***

<https://www.mkyong.com/java/how-to-convert-list-to-set-arraylist-to-hastset/>

<https://www.tutorialspoint.com/java/java_hashset_class.htm>

<http://beginnersbook.com/2013/12/hashset-class-in-java-with-example/>

***Hash Set Examples:***

package arrayss;

import java.util.HashSet;

import java.util.Iterator;

import java.util.Set;

import java.util.TreeSet;

public class HasssetAddingItems {

public static void main(String[] args) {

// TODO Auto-generated method stub

HashSet A=new HashSet();

A.add("s");

A.add("king");

A.add("me");

A.add("l");

System.out.print(A);

System.out.println(A);

//it will clear all the elements in the set

//A.clear();

// Creating a TreeSet of HashSet elements

Set<String> tset = new TreeSet<String>(A);

// Displaying TreeSet elements

System.out.println("TreeSet contains: ");

for(String temp : tset){

System.out.println(temp);}

//Iterating Through every element of the set

Iterator<String> it = A.iterator();

while(it.hasNext()){

System.out.println(it.next());

}

}

}

***write code to add items to integer, string array***

[**http://tutorials.jenkov.com/java/arrays.html**](http://tutorials.jenkov.com/java/arrays.html)

**package** arrayss;

**import** java.util.ArrayList;

**public** **class** AddiigElemntAtendofArray {

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

// **TODO** Auto-generated method stub

// Create an ArrayList that holds references to String

ArrayList<String> names = **new** ArrayList<String>();

// Capacity starts at 10, but size starts at 0

System.***out***.println("initial size: " + names.size() );

// Add three String references

names.add("Amy");

names.add("Bob");

names.add("Cindy");

System.***out***.println("new size: " + names.size() );

// Access and print out the Objects

**for** ( **int** j=0; j<names.size(); j++ )

System.***out***.println("element " + j + ": " + names.get(j) );

}

}

***write code to retrieve items from integer, string array***

***write code to add items to ArrayList collection***

***write code to retrieve items from arraylist (using for each loop***\_

USING FOR EACH LOOP:

**package** arrayss;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**public** **class** UsingForEachloopinArrays {

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

// **TODO** Auto-generated method stub

ArrayList<String> crunchifyList = **new** ArrayList<String>();

// add 4 different values to list

crunchifyList.add("eBay");

crunchifyList.add("Paypal");

crunchifyList.add("Google");

crunchifyList.add("Yahoo");

crunchifyList.forEach((hi)->{

System.***out***.print(hi);

});

}

***LOOPING ARRAY LIST IN JAVA***

package com.mkyong.core;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

public class ArrayListLoopingExample {

public static void main(String[] args) {

List<String> list = new ArrayList<String>();

list.add("Text 1");

list.add("Text 2");

list.add("Text 3");

System.out.println("#1 normal for loop");

for (int i = 0; i < list.size(); i++) {

System.out.println(list.get(i));

}

System.out.println("#2 advance for loop");

for (String temp : list) {

System.out.println(temp);

}

System.out.println("#3 while loop");

int j = 0;

while (list.size() > j) {

System.out.println(list.get(j));

j++;

}

System.out.println("#4 iterator");

Iterator<String> iterator = list.iterator();

while (iterator.hasNext()) {

System.out.println(iterator.next());

}

}

}

***write code to add items HashMap***

<https://www.tutorialspoint.com/java/util/java_util_hashmap.htm>

<http://www.careerbless.com/samplecodes/java/beginners/collections_ds/inertandretrieveMap.php>

**package** arrayss;

**import** java.util.HashMap;

**public** **class** Hashmapadding {

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

// **TODO** Auto-generated method stub

// create hash map

HashMap newmap = **new** HashMap();

// populate hash map

newmap.put(1, "tutorials");

newmap.put(2, "point");

newmap.put(3, "is best");

System.***out***.println("Map value before change: "+ newmap);

// put new values at key 3

String prevvalue=(String)newmap.put(3,"is great");

// check returned previous value

System.***out***.println("Returned previous value: "+ prevvalue);

System.***out***.println("Map value after change: "+ newmap);

}

}

***The Output You can see is:***

Map value before change: {1=tutorials, 2=point, 3=is best}

Returned previous value: is best

Map value after change: {1=tutorials, 2=point, 3=is great}

***write code to retrieve items HashMap***

***Good link for HashMap more methods of writing***

[***http://www.java2novice.com/java-collections-and-util/hashmap/all-keys/***](http://www.java2novice.com/java-collections-and-util/hashmap/all-keys/)

HashMap<Integer,String> First=**new** HashMap<Integer,String>();

First.put(45, "Rahul");

First.put(66, "Hello");

First.put(77, "Kiing");

//First.remove(66);

First.replace(77, "JAi");

//ths is print the whole set of the map

System.***out***.println(First);

//to get particular element

System.out.println(First.get(give key number))

//for Iterating through HashMAP

Set<Integer> k=First.keySet();

//Using the Advanced For loop to retrive each key from the map set

**for**(Integer key:k){

System.***out***.println(key);

}

***Write code to add items to hashset***

**package** arrayss;

**import** java.util.HashSet;

**public** **class** HasssetAddingItems {

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

// **TODO** Auto-generated method stub

HashSet A=**new** HashSet();

A.add(1);

A.add("king");

A.add("me");

A.add(998);

System.***out***.print(A);

}

}

OUT PUT:

[1, king, 998, me]

***Write code to retrieve items to hasset***

<https://www.mkyong.com/java/how-to-convert-list-to-set-arraylist-to-hastset/>

<https://www.tutorialspoint.com/java/java_hashset_class.htm>

<http://beginnersbook.com/2013/12/hashset-class-in-java-with-example/>

***Hash Set Examples:***

package arrayss;

import java.util.HashSet;

import java.util.Iterator;

import java.util.Set;

import java.util.TreeSet;

public class HasssetAddingItems {

public static void main(String[] args) {

// TODO Auto-generated method stub

HashSet A=new HashSet();

A.add("s");

A.add("king");

A.add("me");

A.add("l");

System.out.print(A);

System.out.println(A);

//it will clear all the elements in the set

//A.clear();

// Creating a TreeSet of HashSet elements

Set<String> tset = new TreeSet<String>(A);

// Displaying TreeSet elements

System.out.println("TreeSet contains: ");

for(String temp : tset){

System.out.println(temp);}

//Iterating Through every element of the set

Iterator<String> it = A.iterator();

while(it.hasNext()){

System.out.println(it.next());

}

}

}