package com.pioneercoders.datatypeexample;

public class DatatypeEx {

/\*

\* Java primitive Data Types:-

1. Representing Type of the variables and expressions it means the variable is able to which type of value.

2. Representing how much memory is allocated for variable.

3. Specifies range value of the variable.

There are 8 primitive data types in java

Data Type size(in bytes) Range default values

byte 1 -128 to 127 0

short 2 -32768 to 32767 0

int 4 -2147483648 to 2147483647 0

long 8 –9,223,372,036,854,775,808 to 9 ,223,372,036,854,775,807 0

float 4 -3.4e38 to 3.4e 0.0

double 8 -1.7e308 to 1.7e308 0.0

char 2 0 to 6553 single space

boolean no-size no-range false

\* Syntax:- data-type name-of-variable = value/literal;

Ex:- int a=10;

\*

\*/

//A simple program on datatypes with default values

//im just declare datatype with variables but im not assigned any values we don't assigned values jvm will provide default values based on the datatype

static int i;

static short s;

static byte b;

static long l;

static float f;

static double d;

static boolean bo;

static char ch;

public static void main(String[] args) {

//above variables all are static only we can call directly without creating object we can discuss types of later sections

System.out.println("Datatype Demo");

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

System.out.println("Default value for int data type is "+" "+i);

System.out.println("Default value for short data type is "+" "+s);

System.out.println("Default value for byte data type is "+" "+b);

System.out.println("Default value for long data type is "+" "+l);

System.out.println("Default value for float data type is "+" "+f);

System.out.println("Default value for double data type is "+" "+d);

System.out.println("Default value for boolean data type is "+" "+bo);

System.out.println("Default value for char data type is "+" "+ch);

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

}

}

package com.pioneercoders.operatorsexamples;

public class ArithmeticOperatorEx {

public static void main(String[] args) {

//Arithmetic operators: (+,-,\*, /, %)

int x = 10;

int y = 20;

int result = x + y;

System.out.println("Addition = " + result);

result = x - y;

System.out.println("Subtraction = " + result);

result = x \* y;

System.out.println("Multiplication = " + result);

result = y / x;

System.out.println("Division = " + result);

result = x % 3;

System.out.println("Modulos = " + result);

}

}

package com.pioneercoders.operatorsexamples;

public class BitwiseOperatorEx {

/\*

\* Bitwise operators:

& (AND): If both arguments are true then result is true.

| (OR): if at least one argument is true. Then the result is true.

^ (X-OR): if both are different arguments. Then the result is true.

Example:

System.out.println(true&false);//false

System.out.println(true|false);//true

System.out.println(true^false);//true

\* Boolean complement (!) operator:

This operator is applicable only for boolean types but not for integral types.

\*/

public static void main(String[] args) {

System.out.println(4&5);

/\*

\* 16 8 4 2 1 (binary format)

\* 1 0 0 (binary value for 4)

\* 1 0 1 (binary value for 5)

---------

\* 4&5 1 0 0

\*/

System.out.println(!false);//return true

System.out.println(!true);//return false

}

}

package com.pioneercoders.operatorsexamples;

public class BitwiseShiftOperatorEx {

public static void main(String[] args) {

int number = 8; //0000 1000

System.out.println("Original number : " + number);

//left shifting bytes with 1 position

number = number<<1; //should be 16 i.e. 0001 0000

//equivalent of multiplication of 2

System.out.println("value of number after left shift: " + number);

number = -8;

//right shifting bytes with sign 1 position

number = number>>1; //should be 16 i.e. 0001 0000

//equivalent of division of 2

System.out.println("value of number after right shift with sign: " + number);

number = -8;

//right shifting bytes without sign 1 position

number = number>>>1; //should be 16 i.e. 0001 0000

//equivalent of division of 2

System.out.println("value of number after right shift with sign: " + number);

}

}

package com.pioneercoders.operatorsexamples;

public class ConcatenationOperatorEx {

/\* ConcatenationOperator(+)

\* String concatenation operator:

The only operator which is overloaded in java is “+” operator. Sometime it acts as arithmetic addition operator and some time concatenation operator.

If at least one argument is string type then “+” operator acts as concatenation and if both arguments are number type then it acts as arithmetic addition operator.

\*/

public static void main(String[] args) {

String a="bhaskar";

int b=10,c=20,d=30;

System.out.println(a+b+c+d);//bhaskar102030

System.out.println(b+c+d+a);//60bhaskar

System.out.println(b+c+a+d);//30bhaskar30

System.out.println(b+a+c+d);//10bhaskar2030

}

}

package com.pioneercoders.operatorsexamples;

public class DecremenOperatorEx {

public static void main(String[] args) {

int x=4;

int c= 5;

int a=x--;

int b=--x;

int j=c--;

int i=--c;

System.out.println(" Pre Decrement values "+" "+i+" "+b);

System.out.println(" Post Decrement values "+" "+a+" "+j);

}

}

package com.pioneercoders.operatorsexamples;

public class EqualityOperatorEx {

/\*

\* Equality operator :(==,!=)

We can apply equality operators for every primitive type including boolean type also.

\*/

public static void main(String[] args) {

//it returns true or false

System.out.println(10==10.0);//true

System.out.println('a'==97.0);//true

System.out.println(true==true);//true

System.out.println('a'!='b');//true

}

}

package com.pioneercoders.operatorsexamples;

public class IncrementOperatorEx {

/\*

\* Increment

\* 1.pre increment (y=++x)

\* 2.post increment (y=x++)

\*

\* Decrement

\* 1.pre decrement(y=--x)

\* 2.post decrement(y=x--)

\*

\*

\*

Expression Initial value of x Final value of x Final value of y

Y=++x; 10 11 11

Y=x++; 10 11 10

Y=--x; 10 9 9

Y=x--; 10 9 10

We can apply increment or decrement operator only for variables but not for constant values

\*

\*/

public static void main(String[] args) {

int x=4;

int c= 5;

int a=x++;

int b=++x;

int j=c++;

int i=++c;

System.out.println(" Pre Increment values "+" "+i+" "+b);

System.out.println(" Post Increment values "+" "+a+" "+j);

}

}

package com.pioneercoders.operatorsexamples;

public class InstanceOfOperatorEx {

/\*

\* Instanceof operator:

We can use this operator to check whether the given object is of particular type (or) not.

Syntax:

Example:

Thread t=new Thread();

System.out.println(t instanceof Thread);//true

System.out.println(t instanceof Object);//true

System.out.println(t instanceof Runnable);//true

To use “instanceof” operator compulsory there should be some relationship between argument types (either parent-child (or) child-parent (or) same type) otherwise we will get compile time error saying “inconvertible types”.

in above example object class super class for any class

thread class implements runnable

Note:

\*/

public static void main(String[] args) {

String name = "Java";

//test for same class of string

System.out.println(name instanceof String);

// test for subclass of Object

//Object class is parent class for any class

System.out.println(name instanceof Object);

// test for subclass of an interface

//charSequence is interface for String

System.out.println(name instanceof CharSequence);

}

}

package com.pioneercoders.operatorsexamples;

public class RelationalOperatorsEx {

/\*

\* Relational operator: (<, <=, >, >=)

We can apply relational operators for every primitive type except boolean.

\*/

public static void main(String[] args) {

int x = 10;

int y = 20;

//return true or false

boolean result ;

result = x > y;

System.out.println("x > y " + result);

result = x >= y;

System.out.println("x >= y " + result);

result = x < y;

System.out.println("x < y " + result);

result = x <= y;

System.out.println("x <= y " + result);

}

}

package com.pioneercoders.operatorsexamples;

public class TernaryOperatorEx {

/\*

\* Other conditional operators are ? and : which form a ternary (three operands) in the following form:

result = A ? B : C

This is interpreted like this: if A evaluates to true, then evaluates B and assign its value to the result.

Otherwise, if A evaluates to false, then evaluates C and assign its value to the result.

For short, we can say: If A then B else C. So this is also referred as shorthand for an if-then-else statement.

\*/

public static void main(String[] args) {

int x = 10;

int y = 20;

int result = (x > 10) ? x : y;

System.out.println("result 1 is: " + result);

result = (y > 10) ? x : y;

System.out.println("result 2 is: " + result);

}

}

package com.pioneercoders.statementsexamples;

public class IfStatementExample {

public static void main(String args[]){

boolean b=false;

// initial value is false but im re asssigning value as true it goes to if statement and executes the statement 'hello'

if(b=true)

{

System.out.println("hello");

}else{

System.out.println("hi");

}

}

}

package com.pioneercoders.statementsexamples;

import java.util.Scanner;

public class SwitchStatementEx {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter your age");

int age = sc.nextInt();

//switch example on vote, based on entered age is he eligible to vote or not

switch (age) {

case 18:

System.out.println("Eligible to vote");

break;

case 20:

System.out.println("hello when will u registered ur first vote");

break;

case 17:

System.out.println("u r not eligible to vote ");

default:

break;

}

System.out.println("if ur age is greater than 18 u can vote");

}

}

package com.pioneercoders.statementsexamples;

public class SwitchStatementExample {

/\*

\* switch(expression){

case value :

//Statements

break; //optional

case value :

//Statements

break; //optional

//You can have any number of case statements.

default : //Optional

//Statements

}

\*

\*

\*/

public static void main(String[] args) {

//example on switch case with String based values

char grade = 'C';

switch(grade)

//it checks the grade value if matching grade value available it will execute that switch case

//if no value matches with grade value in switch case it will executes default switch case

{

case 'A' :

System.out.println("Excellent!");

break;

case 'C' :

System.out.println("Well done");

break;

case 'D' :

System.out.println("You passed");

case 'F' :

System.out.println("Better try again");

break;

default :

System.out.println("Invalid grade");

}

System.out.println("Your grade is " + grade);

}

}

package com.pioneercoders.loopexamples;

public class DoWhileLoopEx {

/\*

\* Do-while:

If we want to execute loop body at least once then we should go for do-while.

\*/

/\*

\* Do while loop executes statment until certain condition become false.

\* Syntax of do while loop is

\*

\* do

\* <loop body>

\* while(<condition>);

\*

\* where <condition> is a boolean expression.

\*

\* Please note that the condition is evaluated after executing the loop body.

\* So loop will be executed at least once even if the condition is false.

\*/

public static void main(String[] args) {

int i =0;

//first do while executes and prints value 0 and then go to while based on condition it executes, when x value increments to 5 condition becomes false it comes out of the loop and executes remaining statements

do

{

System.out.println("i is : " + i);

i++;

}while(i < 5);

System.out.println("hello comes out of the loop");

}

}

package com.pioneercoders.loopexamples;

public class EnhancedForLoopEx {

/\*

\* Enhanced for loop in Java:

As of Java 5, the enhanced for loop was introduced. This is mainly used to traverse collection of elements including arrays.

Syntax:

The syntax of enhanced for loop is:

for(declaration : expression)

{

//Statements

}

declaration is a variable and its data type is compatible with array elements we are accessing

\* expression may be array we need to loop through

\* ex:

\* for(int number: numbers){

\* i'm accessing number element one by one in array of numbers

\*

\*

\*

\*

\*

\*

\*

\*

\*/

public static void main(String[] args) {

int [] numbers = {10, 20, 30, 40, 50};

//for each loop iteration mainly for iterating arrays elements

for(int number : numbers ){//number is variable of int data type accessing one by one elements of numbers array

System.out.print( number );

System.out.print(",");

}

System.out.print("\n");

//Iterating String array elements one by one

String [] names ={"James", "Larry", "Tom", "Lacy"};

for( String name : names ) {//name is variable of String datatype accessing one by one elements of Strings array

System.out.print( name );

System.out.print(",");

}

}

}

package com.pioneercoders.loopexamples;

public class ForLoopEx {

/\*

\* For Loop

For loop executes group of Java statements as long as the boolean condition evaluates to true.

For loop combines three elements which we generally use: initialization statement, boolean expression and increment or decrement statement.

For loop syntax

for( <initialization> ; <condition> ; <statement>(increment or decrement) ){

<Block of statements>;

}

The initialization statement is executed before the loop starts. It is generally used to initialize the loop variable.

Condition statement is evaluated before each time the block of statements are executed. Block of statements are executed only if the boolean condition evaluates to true.

Statement is executed after the loop body is done. Generally it is being used to increment or decrement the loop variable.

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*/

public static void main(String[] args) {

//demo program on for loop first it evaluates condition and executes statements and then increments and again check the condition and executes until the condition fails

for(int a=0;a<15;a++){

System.out.println("Value of a is "+a);

System.out.println("\n");//it comes out of the loop if a value increment to 15 it comes out of the loop

}

System.out.println("hello comes out of the for loop if condition fails ");

}

}

package com.pioneercoders.loopexamples;

public class ForLoopIterationEx {

//we can iterate elements either by using for loop or foreach loop (enhanced forloop only in java not in other programming languages like c,c++

public static void main(String[] args) {

int [] numbers = {100, 200, 300, 400, 500};

for(int i=0;i<numbers.length;i++){//for(int i=0;i<5;i++) equivalent to numbers.length

System.out.print(numbers[i]);

System.out.println();

}

System.out.print("----------------------------------------------"+"\n");

String [] names ={"James", "Larry", "Tom", "Lacy"};

for(int j=0;j<names.length;j++){

System.out.println(names[j]);

}

}

}

package com.pioneercoders.loopexamples;

/\*

Infinite For loop Example

This Java Example shows how to create a for loop that runs infinite times

in Java program. It happens when the loop condition is always evaluated as true.

\*/

public class InfiniteForLoopEx {

public static void main(String[] args) {

/\*

\* Its perfectely legal to skip any of the 3 parts of the for loop.

\* Below given for loop will run infinite times.

\*/

for(;;)

System.out.println("Hello");

/\*

\* To terminate this program press ctrl + c in the console. or terminate (red icon) in the eclipse console

\*/

}

}

package com.pioneercoders.loopexamples;

public class WhileLoopEx {

/\*

\* If we want evaluate conditions we will statements like if,if-else,switch case

\* if condition is true it will execute one statement otherwise it will go to else block and execute the other statements

\*

\* But if we want to execute group of statements like multiple no of times ,statements are not suitable then we have alternate is loops (for,for-each,while do-while)

\*

\*

\* While loop: if we don’t know the no of iterations in advance then best loop is while loop:

\*

\*

\* A while loop statement in java programming language repeatedly executes a target statement as long as a given condition is true.

Syntax:

The syntax of a while loop is:

while(Boolean\_expression)

{

//Statements

}

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non zero value.

When executing, if the boolean\_expression result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.

When the condition becomes false, program control passes to the line immediately following the loop

\*

\*

\*

\*

\* Here, key point of the while loop is that the loop might not ever run.

\* When the expression is tested and the result is false,

\* the loop body will be skipped and the first statement after the while loop will be executed.

\*

\*

\*

\*/

public static void main(String[] args) {

int x = 10;

while( x < 20 ) {

System.out.print("value of x : " + x );

x++;

System.out.print("\n");

}

//when x value incremented to 20 condition becomes false while loop comes out and executes rest of the statements

System.out.println("hello loop is terminated ");

}

}

package com.pioneercoders.arrayexamples;

public class ArrayEx {

/\*

\* An array is an indexed collection of fixed number of homogeneous data elements.

The main advantage of arrays is we can represent multiple values with the same name so that readability of the code will be improved.

But the main disadvantage of arrays is:

Fixed in size that is once we created an array there is no chance of increasing or decreasing the size based on our requirement that is to use arrays concept compulsory we should know the size in advance which may not possible always.

We can resolve this problem by using collections.

The maximum allowed array size in java is maximum value of int size [2147483647].

\*/

public static void main(String[] args) {

//here i declared array of int type , assigned values and retreiving based on index values

int [] a= {10,20,30,40,50,60,66};//in arrays index position starts from 0

System.out.println("Length of int [] is"+" "+a.length);//it return size int array

System.out.println("Value of 0th index is "+" "+a[0]);//im getting index position 0 value

System.out.println("Value of 1st index is "+" "+a[1]);

System.out.println("Value of 2nd index is "+" "+a[2]);

System.out.println("Value of 3rd index is "+" "+a[3]);

}

}

package com.pioneercoders.arrayexamples;

public class ArrayIteration {

public static void main(String[] args) {

double[] myList = {1.9, 2.9, 3.4, 3.5};

// Case:1 Print all the array elements using for loop without mention index numbers

System.out.println("Approach one Using normal for loop ");

for (int i = 0; i < myList.length; i++) {

System.out.println(myList[i] + " ");

}

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

/\*

\* Case:2 we can iterate array elements using without using index numbers

The foreach Loops:

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

\* ex:

\* double[] myList = {1.9, 2.9, 3.4, 3.5};

// Print all the array elements

\* for(datatypename variable name:arrayname)

for (double element: myList) {

System.out.println(element);

}

\*

\*/

/\*

\* Example : iteration of array elements using foreach loop introduced in 1.5

\*/

System.out.println("Approach two Using foreach loop applicable only arrays ");

int [] list= {20,30,40,50,60,70};

for(int data:list){ //iterating integer elements without mentioning index numbers

System.out.println(data);

}

}

}

package com.pioneercoders.classandobjecttheory;

/\*

\*

\* Class Vs Object:-

Class is a logical entity it contains logics where as object is physical entity it is representing memory.

Class is used to declare logics for that logics memory is allocated by creation of Object.

Class is blur print it decides object creation without class we are unable to create object.

Based on single class(blue print) it is possible to create multiple objects but every object required memory.

or

Object - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.

Class - A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.

Ex:public class Dog{

String breed;//instance variables

int ageC

static String color;//static variables or class level variables

void barking(){

int sec; local variables

}

void hungry(){

}

void sleeping(){

}

}

\* Note: Generally class will have variables and methods

\* Object will have states and behavior

\*

\* class structure

\*

\* //based on accessibility variables are classified on following types

\*

\* Local variables: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

Instance variables: Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

Class variables: Class variables are variables declared with in a class, outside any method, with the static keyword.

\*

\*

\*

\*/

public class ClassAndObjectTheory {

int i= 20;

int j=30;

/\*Creating an Object:

As mentioned previously, a class provides the blueprints for objects. So basically an object is created from a class. In Java, the new key word is used to create new objects.

There are three steps when creating an object from a class:

Declaration: A variable declaration with a variable name with an object type.

Instantiation: The 'new' key word is used to create the object.

Initialization: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example of creating an object is given below:\*/

public static void main(String []args){

//object creation in java by using new keyword to access instances variables and methods

ClassAndObjectTheory cobj= new ClassAndObjectTheory();

//object is created and memory is allocated for memory for ClassAndObjectTheory class

//access instance variables

System.out.println("Instance Variables are"+" "+cobj.i+" "+cobj.j);

}

}

package com.pioneercoders.classandobjecttheory;

public class TypesOfVariables {

/\*

\* Local variables: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

Instance variables: Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

Class variables: Class variables are variables declared with in a class, outside any method, with the static keyword.

\*

\*

\*/

}

package com.pioneercoders.ConstructorExamples;

/\*

\* Constructor in java is a special type of method that is used to initialize the object.

Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Constructors are used to initialize the instance variables of an object.

\*There are two types of constructors,

1) Default Constructor (provided by compiler).

2) User defined Constructor (provided by user) or parameterized constructor

\*

When we create object by using new operator then constructor execution will be done.

Rules to declare constructor:-

1) Constructor name class name must be same.

2) Constructor is able to take parameters.

3) Constructor not allowed explicit return type (return type declaration not possible).

\* Constructor does not return any value

\*

\* Default Constructor:-

1) If we are not write constructor for a class then compiler generates one constructor for you that constructor is called default constructor. And it is not visible in code.

2) Compiler generates Default constructor inside the class when we are not providing any type of constructor (0-arg or parameterized).

3) The compiler generated default constructor is always 0-argument constructor with empty implementation (empty body).

\*

\*

\*/

public class ConstructorEx {

void message(){

System.out.println("im not providing any constructor during runtime while object creation compliler searching for ConstructorEx() constructor if not there it generates 0-argument constructor with empty implementation ");

}

public static void main(String[] args) {

ConstructorEx conex= new ConstructorEx();//constructor initializes compiler generates 0-argument constructor

conex.message();

}

}

package com.pioneercoders.ConstructorExamples;

/\*

\* Inside the class if we declaring at least one constructor (either 0-arg or parameterized) the compiler won’t generate default constructor.

\*/

public class ConstructorExOne {

//im creating parameterized constructor with some parameters

ConstructorExOne(int i) {

System.out.println("Paramerized constructor with integer argument it is assigned while creation of object "+i);

}

public ConstructorExOne(String str,int i) {

System.out.println("Parameterized constructor with parameters String ,integer argument these are assigned while creation of object "+str+" "+i);

}

public static void main(String[] args) {

ConstructorExOne conex= new ConstructorExOne(27);

ConstructorExOne conex1= new ConstructorExOne("sri",100);

}

}

package com.pioneercoders.ConstructorExamples;

public class DefaultConstructorEx {

//instance variables

int eid;

String ename;

double esal;

void display()

{ //printing instance variables values

System.out.println("\*\*\*\*Employee details\*\*\*\*");

System.out.println("Employee name :-->"+ename);

System.out.println("Employee eid :-->"+eid);

System.out.println("Employee sal :-->"+esal);

}

public static void main(String[] args)

{ // during object creation 0-arg cons executed then values are assigned

DefaultConstructorEx defaultvalues = new DefaultConstructorEx();

defaultvalues.display();

}

/\*

\* Note: - in above example during object creation time default constructor is executed with empty implementation and initial values of instance variables (default values) are printed .

\*/

}

package com.pioneercoders.ConstructorExamples;

public class ParameterizedConstructorEx {

//instance variables

int eid;

String ename;

double esal;

ParameterizedConstructorEx()

{ //assigning values to instance values during object creation

eid=111;

ename="ratan";

esal =60000;

}

void display()

{ //printing instance variables values

System.out.println("\*\*\*\*Employee details\*\*\*\*");

System.out.println("Employee name :-->"+ename);

System.out.println("Employee id :-->"+eid);

System.out.println("Employee esal :-->"+esal);

}

public static void main(String[] args) {

ParameterizedConstructorEx pcon= new ParameterizedConstructorEx();

// during object creation 0-arg cons executed then values are assigned

//during object creation it invokes parameterized constructor 0-argument(ParameterizedConstructor) invokes

pcon.display();

}

}

package com.pioneercoders.thiskeywordexamples;

public class ThisEx {

/\*

\* this keyword:-

this keyword is holding current class reference variable and it is used to represent,

a. Current class variables.

b. Current class methods.

c. Current class constructors.

Current class variables:-

This keyword not required:-

\*

\*/

//instance variables

int a=100;

int b=200;

void add(int i,int j)//local variables

{ System.out.println(a+b);//instance variables addition

System.out.println(i+j);//local variables addition

}

public static void main(String[] args)

{ ThisEx t = new ThisEx();

t.add(10,20);

}

//In above example instance variables and local variables having different names so this keyword not required.

//if instance and local variables are same we have to go for this keyword

}

package com.pioneercoders.thiskeywordexamples;

public class ThisExOne {

//This keyword required:-

//instance variables

int a=100;

int b=200;

void add(int a,int b)//local variables

{

System.out.println(a+b);//local variables addition

System.out.println(this.a+this.b);//instance variables addition

// here local variables and instance variables are same to access instance variables we have to use this keyword

}

public static void main(String[] args)//static method

{ ThisExOne t = new ThisExOne();

t.add(10,20);

}

//In above example instance variables and local variables having same name at that situation we are able to print local variables directly but to represent instance variables use this keyword.

}

package com.pioneercoders.thiskeywordexamples;

public class ThisKeywordWithParameterized {

int id;

String name;

public ThisKeywordWithParameterized(int id,String name)

{

//instance variables and local variables are same to access current class instance variables we have to use this keyword

this.id=id;

this.name=name;

}

void display(){

System.out.println("values are" +id+" "+name);

}

public static void main(String[] args) {

ThisKeywordWithParameterized key = new ThisKeywordWithParameterized(20,"srinadh");

ThisKeywordWithParameterized key1 = new ThisKeywordWithParameterized(30,"rajesh");

key.display();

key1.display();

}

}

package com.pioneercoders.abstractionandencapsulation;

/\*

\* The process highlighting the set of services and hiding the internal implementation is called abstraction.

1.Bank ATM Screens Hiding the internal implementation and highlighting set of services like , money transfer, mobile registration,…etc).

2. When we see a nice car on the road as a casual onlooker, we get to see the whole picture. The car as a one single unit, a vehicle. We do not see the underlying complex mechanical engineering.

we cannot bother about internal working of it

We are achieving abstraction concept by using Abstract classes & Interfaces.

Points to Remember

Abstraction is process of hiding the implementation details and showing only the functionality.

Abstraction in java is achieved by using interface and abstract class. Interface give 100% abstraction and abstract class give 0-100% abstraction.

A class that is declared as abstract is known as abstract class.

Syntax:

abstract class <class-name>{}

An abstract class is something which is incomplete and you cannot create instance of abstract class.

If you want to use it you need to make it complete or concrete by extending it.

A class is called concrete if it does not contain any abstract method and implements all abstract method inherited from abstract class or interface it has implemented or extended.

A method that is declare as abstract and does not have implementation is known as abstract method.

If you define abstract method than class must be abstract.

Syntax:

abstract return\_type method\_name ();

An abstract method in Java doesn't have body, it’s just a declaration. In order to use abstract method you need to override that method in Subclass.

\* An abstract class will have abstract and non-abstract methods

\* it is have atleast one abstract method completely make that class as abstract

\*/

// Abstraction Without abstract method

class Employee extends Person {

private String empCode;

public String getEmpCode() {

return empCode;

}

public void setEmpCode(String empCode) {

this.empCode = empCode;

}

}

abstract class Person {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

public class AbstractionMain{

public static void main(String args[]){

//INSTIATING AN ABSTRACT CLASS GIVES COMPILE TIME ERROR

//Person p = new Person() ;

//THIS REFERENCE VARIABLE CAN ACESS ONLY THOSE METHOD WHICH ARE OVERRIDDEN

Person person = new Employee();

//we are assigning Abstract class reference to overidden method

Employee emp = new Employee();

person.setName("Gopi");

System.out.println(person.getName());

}

}

package com.pioneercoders.abstractionandencapsulation;

public class EncapsulationEx {

/\*

\* Encapsulation:-

The process of binding the data and code as a single unit is called encapsulation.

We are able to provide more encapsulation by taking the private data(variables) members.

If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class. we have to use setters and getters which are public methods that will give access to outside members

To get and set the values from private members use getters and setters to set the data and to get the data.

However if we setup public getter and setter methods to update (for e.g. void setSSN(int ssn))and read (for e.g. int getSSN()) the private data fields then the outside class can access those private data fields via public methods. This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes. That’s why encapsulation is known as data hiding.

\*/

private int ssn;

private String empName;

private int empAge;

//getter methods are called accessor methods these methods will return some data

public int getSsn() {

return ssn;

}

//setter methods are called mutator methods these are used to set the data ,these only sets data it will not return any values

public void setSsn(int ssn) {

this.ssn = ssn;

}

public String getEmpName() {

return empName;

}

public void setEmpName(String empName) {

this.empName = empName;

}

public int getEmpAge() {

return empAge;

}

public void setEmpAge(int empAge) {

this.empAge = empAge;

}

}

package com.pioneercoders.abstractionandencapsulation;

public class EncapsulationTest {

public static void main(String[] args) {

EncapsulationEx obj = new EncapsulationEx();

obj.setEmpName("Mario");

obj.setEmpAge(32);

obj.setSsn(112233);

System.out.println("Employee Name: " + obj.getEmpName());

System.out.println("Employee SSN: " + obj.getSsn());

System.out.println("Employee Age: " + obj.getEmpAge());

}

}

package com.pioneercoders.overridingex;

/\*

\* Rules overriding

\* Runtime polymorphism [Method Overriding]:-

\*

1) If we want to achieve method overriding we need two class with parent and child relationship.

2) The parent class method contains some implementation (logics).

a. If child is satisfied use parent class method.

b. If the child class not satisfied (required own implementation) then override the method in Child class.

3) A subclass has the same method as declared in the super class it is known as method overriding.

The parent class method is called ===> overridden method

The child class method is called ===> overriding method

While overriding methods must fallow these rules:-

1) While overriding child class method signature & parent class method signatures must be same otherwise we are getting compilation error.

2) The return types of overridden method & overriding method must be same.

3) While overriding check the modifiers permission like the sub class method modifier is having same permission or increasing level of parent class method bur not decreasing order.

4) You are unable to override final methods. (Final methods are preventing overriding).

5) While overriding check the covariant-return types.

6) Static methods are bounded with class hence we are unable to override static methods.

\*/

public class BaseClass {

public void methodToOverride() //Base class method

{

System.out.println ("I'm the method of BaseClass");

}

public class DerivedClass extends BaseClass

{

public void methodToOverride() //Derived Class method

{

System.out.println ("I'm the method of DerivedClass");

}

}

public static void main(String[] args) {

// BaseClass reference and object

BaseClass obj1 = new BaseClass();

// BaseClass reference but DerivedClass object

BaseClass obj2 = obj1.new DerivedClass();

// Calls the method from BaseClass class

obj1.methodToOverride();

//Calls the method from DerivedClass class

obj2.methodToOverride();

}

}

package com.pioneeroders.overloadingex;

public class OverLoadingEx {

/\*

\* Compile time polymorphism [Method Overloading]:-

1) If java class allows two methods with same name but different number of arguments such type of methods are called overloaded methods.

2) We can overload the methods in two ways in java language

a. By passing different number of arguments to the same methods.

void m1(int a){ }

void m1(int a,int b){ }

b. Provide the same number of arguments with different data types.

void m1(int a){ }

void m1(char ch){ }

3) If we want achieve overloading concept one class is enough.

4) It is possible to overload any number of methods in single java class.

\*/

void demo (int a)

{

System.out.println ("a: " + a);

}

void demo (int a, int b)

{

System.out.println ("a and b: " + a + "," + b);

}

double demo(double a) {

System.out.println("double a: " + a);

return a\*a;

}

public static void main (String args [])

{

OverLoadingEx Obj = new OverLoadingEx();

double result;

Obj.demo(100);

Obj .demo(10, 20);

result = Obj .demo(5.5);

System.out.println("O/P : " + result);

}

}

package com.pioneeroders.finalkeyword;

public class FinalBike{

final void m1(){

System.out.println("running");

}

class Engine extends FinalBike{

// we cannot override final methods

void m1(){

System.out.println("running with 100 kmph");

}

}

public static void main(String[] args) {

FinalBike b= new FinalBike();

b.m1();

Engine eg= b.new Engine();

eg.m1();

//if we override final methods we get

/\*

\* Exception in thread "main" java.lang.VerifyError: class com.corejavaex.Bike$Engine overrides final method m1.()V

at java.lang.ClassLoader.defineClass1(Native Method)

at java.lang.ClassLoader.defineClass(ClassLoader.java:800)

at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)

at java.net.URLClassLoader.defineClass(URLClassLoader.java:449)

at java.net.URLClassLoader.access$100(URLClassLoader.java:71)

at java.net.URLClassLoader$1.run(URLClassLoader.java:361)

at java.net.URLClassLoader$1.run(URLClassLoader.java:355)

at java.security.AccessController.doPrivileged(Native Method)

at java.net.URLClassLoader.findClass(URLClassLoader.java:354)

at java.lang.ClassLoader.loadClass(ClassLoader.java:425)

at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)

at java.lang.ClassLoader.loadClass(ClassLoader.java:358)

at sun.launcher.LauncherHelper.checkAndLoadMain(LauncherHelper.java:482)

\*/

}

}

package com.pioneeroders.finalkeyword;

/\*

\* It makes a class final, meaning that the class can not be inheriting by other classes. When we want to restrict inheritance then make class as a final.

if we want to restrict inheritance functionality simple we make class as final nobody can extends our class

\*/

final class FinalClassEx {

void message(){

System.out.println("hii");

}

class MyClass extends FinalClassEx{

void message() {

System.out.println("hello");

}

}

public static void main(String[] args) {

FinalClassEx fc= new FinalClassEx();

MyClass mc= fc.new MyClass();

/\*

\* Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The type MyClass cannot subclass the final class FinalClassEx

at com.corejavaex.FinalClassEx$MyClass.<init>(FinalClassEx.java:7)

at com.corejavaex.FinalClassEx.main(FinalClassEx.java:15)

\*/

}

}

package com.pioneeroders.finalkeyword;

public class FinalEx {

/\*

\* The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1.variable

2.method

3.class

\*/

/\*

\* Java final variable

\* In previous programming languages if values are standard those values are not changed we can declare those values as constants ex pi=3.14 it is constant

\* it will never change to represent constant values in c we have keyword const

\* ex:const float pi=3.14;

\* but in java language to refer we have final keyword to refer constants

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Points to Remember:

1) A constructor cannot be declared as final.

2) Local final variable must be initializing during declaration.

3) All variables declared in an interface are by default final.

4) We cannot change the value of a final variable.

5) A final method cannot be overridden.

6) A final class not be inherited.

7) If method parameters are declared final then the value of these parameters cannot be changed.

\*/

}

package com.pioneeroders.finalkeyword;

public class FinalVariableEx {

//if nobody want to change the values of our variables we can restrict easily by making that variable as final

//they can use our variables but they cannot change it

final float pi=3.14f;

void m1(){

pi=3.6f;

}

public static void main(String[] args) {

FinalVariableEx ex= new FinalVariableEx();

ex.m1();

}

//if u run the above we will complilation error

/\*

\* Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The final field FinalVariableEx.pi cannot be assigned

at com.corejavaex.FinalVariableEx.m1(FinalVariableEx.java:6)

at com.corejavaex.FinalVariableEx.main(FinalVariableEx.java:10)

\*/

}

package com.pioneeroders.statickeyword;

public class StaticExample {

/\*

\* The variables which are declared inside the class and outside of the methods with static modifier is called static variables.

Static variables are visible all methods and constructors of a particular class. Call the static members by using class Name.

Static variables memory allocated at the time of .class file loading and memory released at .class file unloading time.

Static variables are stored in non-heap memory.

We can Directly static variables and methods without creating object

\*/

/\*

\* Calling of static variables:-

We are able to access the static members inside the static area in three ways.

a. Directly possible.

b. By using class name.

c. By using reference variable.

\*/

static int a=10;

public static void main(String[] args) {

System.out.println(a); //1-way(directly possible)

System.out.println(StaticExample.a); //2-way(By using class name)

StaticExample t=new StaticExample();

System.out.println(t.a); //3-way(By using reference variable)

}

}

package com.pioneeroders.Inheritanceprograms;

/\*

\* Inheritance:-

1. The process of acquiring fields(variables) and methods(behaviors) from one class to another class is called inheritance.

2. The main objective of inheritance is code extensibility whenever we are extending class automatically the code is reused.

3. In inheritance one class giving the properties and behavior & another class is taking the properties and behavior.

4. Inheritance is also known as is-a relationship. By using extends keyword we are achieving inheritance concept.

5. extends keyword used to achieve inheritance & it is providing relationship between two classes when you make relationship then able to reuse the code.

6. In java parent class is giving properties to child class and Child is acquiring properties from Parent.

7. To reduce length of the code and redundancy of the code sun people introduced inheritance concept

8.Parent class is called super or Base class

9.Child class is called sub class or derived class

10.Main advantage is code reusability

if parent class has add method , if we want to perform arithmetic operations we again don't create add method in our class simply we use same add method in our class by using extends keyword we can use parent class add method

\*/

class ParentInheritanceEx {//super class

public int add(int a,int b){

int x= a+b;

return x;

}

class Child extends ParentInheritanceEx{//sub class

//we need add method again we dont create just inheriting from ParentInheritanceEx class

public int mul (int x,int y){

int z= x\*y;

return z;

}

}

public static void main(String[] args) {

ParentInheritanceEx p= new ParentInheritanceEx();

Child ch=p. new Child();

int output= ch.mul(10, 23);

System.out.println(output);

int result= ch.add(23, 21);

System.out.println(result);

}

}

package com.pioneeroders.polymorphismexamples;

public class PolymorphismEx {

/\*

\* Polymorphism:-

One thing can exhibits more than one form is called polymorphism.

Polymorphism shows some functionality(method name same) with different logics execution.

The ability to appear in more forms is called polymorphism.

Polymorphism is a Greek word poly means many and morphism means forms.

There are two types of polymorphism in java

1) Compile time polymorphism / static binding / early binding

[method execution decided at compilation time]

Example :- method overloading.

2) Runtime polymorphism /dynamic binding /late binding.

[Method execution decided at runtime].

Example :- method overriding.

\*/

}

package com.pioneercoders.interfaceexamples;

public class InterfaceAndAbstractClassDifference {

/\*

\* interface AbstractClass

\*

\* 1) If we don’t’ know anything about implementation 1) If we are talking about implementation but not completely

\* just we have requirement specification (partial implementation) then we should

\* then we should go for interface. go for abstract class.

\*

\* 2)Every method present inside interface 2)Every method present inside abstract

\* is always public and abstract whether class need not be public and abstract

\* we are declaring or not.

\*

\* 3)We can’t declare interface methods 3) There are no restrictions on abstract

\* with the modifiers private, protected, class method modifiers.

\* final, static, synchronized, native,

\* strictfp

\*

\* 4)Every interface variable is always 4) Every abstract class variable need not

\* public static final whether we are be public static final

\* whether we declaring or not

\*

\* 5)For the interface variables compulsory 5) It is not require to perform

\* we should perform initialization at the initialization for abstract class variables

\* time of declaration otherwise we will at the time of declaration.

\* get compile time error

\*

\* 6) Inside interface we can’t take static 6) Inside abstract class we can take both

\* and instance blocks. static and instance blocks.

\*

\*

\*

\*/

}

package com.pioneercoders.interfaceexamples;

public class InterfaceTheory {

/\*

\*

\* Genearal definition interface is mediator between client and service provider

\*

\* Def1: Any service requirement specification (srs) is called an interface.

Example1: Sun people responsible to define JDBC API and database vendor will provide implementation for that.

\* Sun

\* |

\* JDBC API

\* |

\*

\* 1.oracle driver 2.mysql driver 3.db2 driver

\*

\* Def2: Sun people define SERVLET API to develop web applications web server vendor is responsible to provide implementation.

\*

\* sun

\* |

\* servlet api

\* |

\* 1. tomcat server 2.websphere server 3.jboss server

\*

\* Def3: From the client point of view an interface define the set of services what his excepting. From the service provider point of view an interface defines the set of services what is offering. Hence an interface is considered as a contract between client and service provider.

\*

\* Example: ATM GUI screen describes the set of services what bank people offering, at the same time the same GUI screen the set of services what customer his excepting hence this GUI screen acts as a contract between bank and customer.

\*

\* Declaration and implementation of an interface:

Note1: Whenever we are implementing an interface compulsory for every method of that interface we should provide implementation otherwise we have to declare class as abstract in that case child class is responsible to provide implementation for remaining methods.

Note2: Whenever we are implementing an interface method compulsory it should be declared as public (to provide implementation by service providers) otherwise we will get compile time error.

Note3:100% pure abstract classes is considered as an interface.

\*

\* we can provide implementation for interfaces by using implements keyword

\*

\* it means im implementing ur interface and i will provide service or implementation logic to u

\* in interface all methods are abstract

\* we can extend only one class but we can implement multiple interfaces

\*/

}

package com.pioneercoders.interfaceexamples;

public interface MathematicalOperationInterface {

//whether we declare all methods present inside interface public and abstract only if we make method as private nobody can access and provide implementation for it ,because we cannot access private methods outside the class

public int add(int a,int b);

public int sub(int c,int d);

public int mul(int x,int y);

public int div(int i, int j);

}

package com.pioneercoders.interfaceexamples;

//if we implement any interface compulsory we have to provide implementation for that

public class MathematicalOperationInterfaceImplementation implements MathematicalOperationInterface{

@Override

public int sub(int c, int d) {

int subresult = c-d;

// TODO Auto-generated method stub

return subresult;

}

@Override

public int mul(int x, int y) {

// TODO Auto-generated method stub]

int mulresult= x\*y;

return mulresult;

}

@Override

public int div(int i, int j) {

// TODO Auto-generated method stub

int divresult = i/j;

return divresult;

}

@Override

//im providing implementation for add method

public int add(int a, int b) {

int c= a+b;

// TODO Auto-generated method stub

return c;

}

public static void main(String[] args) {

//we cannot create object for interface ,im providing implementation for it by creating implementation class

MathematicalOperationInterface mathimpl= new MathematicalOperationInterfaceImplementation();

int result = mathimpl.add(20, 37);

System.out.println("Addition"+" "+result);

int subtractresult = mathimpl.sub(43, 22);

System.out.println("Subtraction"+" "+subtractresult);

int multiplyresult= mathimpl.mul(20, 12);

System.out.println("Multiply "+" "+multiplyresult);

int divisionresult= mathimpl.div(30, 2);

System.out.println("Division"+" "+divisionresult);

//Note If any methods common for all classes or most of the class we will place them in one interface

}

}

//package com.pioneercoders.abstractclassexamples;

public abstract class AbstractEx {

public void message(){

System.out.println("hello msg ");

}

abstract void shape();

abstract void draw();

abstract int add(int a,int b);

}

package com.pioneercoders.abstractclassexamples;

public class AbstractExImpl extends AbstractEx {

@Override

void shape() {

System.out.println("im drawing s shape ");

// TODO Auto-generated method stub

}

@Override

void draw() {

System.out.println("im drawing cirlce ");

// TODO Auto-generated method stub

}

@Override

int add(int a, int b) {

// TODO Auto-generated method stub

int result = a+b;

return result;

}

public static void main(String[] args) {

//we are calling AbstractEx class methods but actually they are not providing implementations end user will not who is providing implementation so in this we can achieve abstraction using abstract class

AbstractEx abs = new AbstractExImpl();

abs.shape();

abs.draw();

abs.message();

//similarity between interface and abstract class we cannot create objects for both interfaces and abstract class

//we can provide implementation for interfaces by using implements keyword for abstract class by using extends keyword

//we can implements any number of interfaces at a time but we can extend multiple abstract classes at a time (multiple inheritance not possible )

//we have only abstract methods in interfaces but in abstract classes we can have both instance and abstract methods

//we don't any idea about implementation our choice is interface , but we know partial implementation(some instance methods ) our choice is abstract classes

int res = abs.add(21, 26);

System.out.println(res);

}

}

package com.pioneercoders.abstractclassexamples;

public class AbstractTheory {

/\*

\* What is the difference between interface, abstract class and concrete class?

When we should go for interface, abstract class and concrete class?

If we don’t know anything about implementation just we have requirement specification then we should go for interface.

If we are talking about implementation but not completely (partial implementation) then we should go for abstract class.

If we are talking about implementation completely and ready to provide service then we should go for concrete class.

\*

\* If we know partial implementation then we should go for abstract class

\* If a class contains abstract methods compulsory make that class as abstract

\* There are no restrictions on abstract

\* It can have abstract and non-abstract methods

\* we can achieve abstraction functionality using abstract class only

\* we can't create object for abstract class if we want to provide implementation for abstract classes by using extends keyword

\*

\*/

}

package com.pioneercoders;

public class ExceptionEx {

public static void main(String[] args) {

/\*

\* The exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

\* The core advantage of exception handling is to maintain the normal flow of the application. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

statement 9;

statement 10;

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

\*

//we can handle only exception but errors if error occur program could be terminated in the middle or sometimes we need start our system also

Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

Checked Exception

Unchecked Exception

Error

Difference between checked and unchecked exceptions

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Generally all filerelated exceptions are checked only, because compliler forces to throw filenotfoundexception whether file is there are not it doesnt matter

and Database related SqlException also

\*/

//Common scenarios where exceptions may occur

//There are given some scenarios where unchecked exceptions can occur. They are as follows:

// Scenario where ArithmeticException occurs

//If we divide any number by zero, there occurs an ArithmeticException.

int a=50/0;//ArithmeticException

// Scenario where NullPointerException occurs

//If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

String s=null;

System.out.println(s.length());//NullPointerException

//3) Scenario where NumberFormatException occurs

//The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

String str="abc";

int i=Integer.parseInt(s);//NumberFormatException

//4) Scenario where ArrayIndexOutOfBoundsException occurs

//If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

int a1[]=new int[5];

a1[10]=50; //ArrayIndexOutOfBoundsException

}

}

package com.pioneercoders;

import java.lang.Class;

import java.lang.String;

import java.lang.System;

public class langex {

public static void main(String[] args) {

System.out.println("default");

}

}

package com.pioneercoders;

public class TryCatchEx {

//first try without try catch

public static void main(String[] args) {

int result =50/0;

System.out.println(result);

System.out.println("remaining code");

System.out.println("has to print");

//if we run without handling exception

//remaining code will not be executed

//its better to write using try catch

//Java try block is used to enclose the code that might throw an exception. It must be used within the method.

//Java try block must be followed by either catch or finally block.

/\*Syntax of java try-catch

try{

code that may throw exception

}catch(Exception\_class\_Name ref){}

\*

\*Syntax of try-finally block

try{

//code that may throw exception

}finally{}

\*

\*Java catch block is used to handle the Exception. It must be used after the try block only.

\*

\*/

// after this refer example TryCatchExampleone in that program division is handled

//using try catch

}

}

package com.pioneercoders;

public class TryCatchExampleone {

public static void main(String[] args) {

//division with try catch

try{

int data=50/0;

//while during division it may throw exception

//catch will handle exception will exception comes

//and prints remaining code as it is

}catch(ArithmeticException e){System.out.println(e);}

System.out.println("remaining code");

System.out.println("has to print");

//if we get any exception also program will not get any abnormal termination,it prints the exception statements if we handle those exceptions remaining statements will executes without any problem

}

}

package com.pioneercoders;

import java.io.File;

import java.io.IOException;

public class FileCreate {

public static void main(String[] args) throws IOException {

File f = new File("D:/createfile.txt");

if(f.exists()){//if file created it terurns true

System.out.println("created already");

}else{//if file is not created it creates

f.createNewFile();

System.out.println("justnow created");

}

//if we run first time fielis not created previously that why it created file and print statement

//justnow created

//if we run second time the file is created already and prints created already statement

}

}

package com.pioneercoders;

/\* Utilizing the same resource or same memory for executing the applications or programs so that we can reduce the time

\* Multithreading in Java

Multithreading in java is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

Advantage of Java Multithreading

1) It doesn't block the user because threads are independent and you can perform multiple operations at same time.

2) You can perform many operations together so it saves time.

3) Threads are independent so it doesn't affect other threads if exception occur in a single thread.

\*

\*

\* A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.

Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.

\* threads are smallest parts in process

\* thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS and one process can have multiple threads.

\*/

//in every program internally to execute our program threads are running

//in java also to run the main method internally some threads are running along with main thread

//some child threads also running including main thread

//in our program if we want to run multiple threads at a time

//we can define in two ways either by implementing 1.Runnable interface

// 2.by extends thread class

//Defining a Thread by extending “Thread class”:

public class MyThread extends Thread {

/\* public void run(){

for(int i=0;i<10;i++){

System.out.println("child thread ");

}

\*/

public static void main(String[] args)

{

MyThread t=new MyThread();//Instantiation of a Thread

//to start the child we have to call start() method

//internally start method will have run()

/\* start method internal code

\*

\* start{

\* public void run(){

\* //no implementation

\* }

\*/

t.start();//starting of a Thread

//after calling start method it checks whether we overidden run() method or not

//if not overridden it calls start and its run() method but it has empty

//implementation in this program we overidden run() method

//we cannot expect multithreading output it changes from system to system based on operating system

for(int i=0;i<20;i++)

{

System.out.println("main thread");

}

}

}

package com.pioneercoders.LegacyClasses;

import java.io.FileReader;

import java.io.IOException;

import java.util.Properties;

public class PropertiesRead {

public static void main(String[] args) throws IOException {

//to read any data from the files we have to use files streams by using filereader read the data

FileReader reader= new FileReader("E:/mydatabaseprop.properties");

//create the properties file

Properties props= new Properties();

//properties file have to load the reader the file reader already reads the data

props.load(reader);

//we have to get the values from propeties file by using key values only

System.out.println(props.getProperty("name"));//name is keyname

System.out.println(props.getProperty("password"));//password is keyname

}

}

package com.pioneercoders.LegacyClasses;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Properties;

public class PropertiesWrite {

public static void main(String[] args) throws IOException {

// in this we are explaining how to write data to properties file

//if we write data in properties file we can change modifications easily we cannot complile java code

//if we made changes in properties file

//first set the data to properties file

//properties file stores the data in the format of key,values(like HashMap)

Properties props= new Properties();

props.setProperty("name", "srinadh");

props.setProperty("password", "code");

//after writing store the data to file using filewriter call store method

// 1.to write data to properties file use store()method

// 2.to read properties from properties file use load() method

props.store(new FileWriter("E:/mydatabaseprop.properties"), "these are database details");

}

}

package com.pioneercoders.LegacyClasses;

import java.util.Stack;

public class StackExample {

public static void main(String[] args) {

//it is the child class of Vector

//All list implementation classes allows duplicates

Stack stack = new Stack();

System.out.println(stack.isEmpty());//returns true because stack will not have elements

//add elements to the stack using push()

stack.add("sri");//first element stored last (index 2)

stack.add("mani");//index 1

stack.add("anand");//(index 0 )//stored first in stack //it follows LastInFirstOut mechanism

System.out.println(stack);

stack.pop();//removes last inserted element anand and returns top of the stack

System.out.println(stack);

stack.peek();

System.out.println(stack);//it removes nothing and returns top of the stack

stack.search("sri");

//returns index value of sri value

System.out.println(stack.search("sri"));

System.out.println(stack.search("rajesh"));//if search value is not there in stack it returns -1

}

}

package com.pioneercoders.LegacyClasses;

import java.util.Vector;

public class VectorEx {

public static void main(String[] args) {

//introduced in 1.0 version (legacy classes)

//it also allows duplicates,null,heterogenenous elements

Vector vec= new Vector<>();

//initial capacity is 10

System.out.println(vec.capacity());

for(int i=0;i<10;i++){

vec.add(i);

//added values to the vector

System.out.println(i);

}

//vector will have 10 values im adding 11 th element

vec.add("sri");

//vector has reached the initial capacity it increases it size automatically upto 2\*intialcapacity(20)

//lets seee the values and capacity after adding 11th element

System.out.println(vec);

System.out.println(vec.capacity());

}

}

package com.pioneercoders.Listimplementations;

import java.util.ArrayList;

public class ArrayListEx {

public static void main(String[] args) {

//collections are like arrays only but some differences are there we can increase

//array size once created size ,we cannot increase in arrays but collection it will increase automatically

//in arrays it stores only homogeneous data means if we declare array as int type we can store only integers but in collections we can store \

//heterogenoues data also it means we can store integers,null, string,boolean values in single collection class

//every collection class or interface internally they are using some data structures

//in this we are showing example on Arraylist

//First create the arraylist

ArrayList li= new ArrayList();

li.add(1);

li.add("srinadh");

li.add(null);

li.add(true);

System.out.println(li);

//if u observe the output arraylist allows null,boolean int,String

//and insertion order is preserved also

//it allows dulpicates also

li.add(1);

System.out.println(li);

}

}

package com.pioneercoders.Listimplementations;

import java.util.ArrayList;

import java.util.Iterator;

public class ArrayListIteration {

/\*

\* Retrieving objects of collections classes:-

We are able to retrieve the objects from collection classes in 3-ways

1) By using for-each loop.

2) By using cursors.

3) By using get() method.

\*/

public static void main(String[] args) {

//ArrayList able to store only String Objects

ArrayList<String> al =new ArrayList<String>();

al.add("A");

al.add("B");

al.add("C");

al.add("D");

al.add(null);

//1st appraoch to print Collection class elements (by using for-each loop)

for (String a : al )

{

System.out.println(a);

}

//2nd approach to print Collection class elements (by using cursors)

Iterator itr1 = al.iterator(); //normal version of Iterator

while (itr1.hasNext())

{ String str =(String)itr1.next(); //type casting required because normal version

System.out.println(str);

}

Iterator<String> itr2 = al.iterator(); //generic version of Iterator

while (itr2.hasNext())

{ String str =itr2.next(); //type casting not required because generic version

System.out.println(str);

}

//3rd approach to print objects by using get() method

int size = al.size();

for (int i=0;i<size;i++)

{ System.out.println(al.get(i));

}

}

}

package com.pioneercoders.Listimplementations;

import java.util.ArrayList;

import java.util.Iterator;

public class IteratorEx {

public static void main(String[] args) {

//we have to use cursors to iterate collection elements one by one

//we are using iterator to iterate values of arraylist

//create the arraylist and add the values

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

li.add(1);

li.add(11);

li.add(13);

li.add(17);

li.add(19);

li.add(12);

li.add(11);

// create the iterator and which collection element we are iterating

Iterator itr= li.iterator();

//hasnext returns true if next element is there

while(itr.hasNext()){

//we are iterating integer values

Integer values =(Integer)itr.next();

//next method iterates one by one value

System.out.println(values);

System.out.println(li);

}

}

}

package com.pioneercoders.Listimplementations;

import java.util.LinkedList;

public class LinkedListEx {

public static void main(String[] args) {

//it is one of the implementation of List interface

//it is better if our requirement is insertion or deletion in the middle

LinkedList li= new LinkedList();

li.add(1);

li.add("srinadh");

li.add(null);

li.add(true);

System.out.println(li);

//adding first and last elements

li.addFirst("gopi");

li.addLast("anand");

System.out.println(li);

//deleting elements

li.removeFirst();

li.removeLast();

System.out.println(li);

}

}

package com.pioneercoders.mapimplementations;

public class Emp {

int eid;

String ename;

Emp(int eid,String ename)

{this.eid=eid;

this.ename=ename;

}

}

package com.pioneercoders.mapimplementations;

import java.util.Collection;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import java.util.Set;

public class HashMapEx {

public static void main(String[] args) {

//Map also similar to collections but it stores

//values in the form of key and values

//underlying datastrucutre is hashtable

//map initial capacity is 16

//one key -value pair is called entry

Map m= new HashMap();

//we are not adding element

System.out.println(m.size());//but initial capacity

//is 16

//if i want to check the map is empty r not we can use

//the following method

System.out.println(m.isEmpty());//returns true map has no data

//add element and again check

//for adding we have to use put()

m.put("name", "name");

System.out.println(m.isEmpty());//map has values

System.out.println(m.size());

//duplicates keys not allowed but duplicate values any no of times if key is already thier new key added in place of new key with new value

//lets check once key=name already thier im adding again but changing value

m.put("name", "srinadh");

System.out.println(m);//key updated with new value

//null also allowed for keys but only once but for values any no of times

m.put(null, null);

System.out.println(m);

m.put("age", 22);

System.out.println(m);

//map doesnt follows insertion order

//to get map data we have three methods

//1.to get only keys we have to use keyset

//method set s =m.keyset

Set s= m.keySet();

System.out.println(s);

//2 to get keys and values we have to use .values()

//Collection c= m.values();

Collection c= m.values();

System.out.println(c);

//3 to get keys and values we have to use entrySet()

//Set s1=m.entrySet();

Set s1=m.entrySet();

System.out.println(s1);

//to iterate map values first set the map values using entrySet

Iterator itr = s1.iterator();

while(itr.hasNext()){

//iterate map values using iterator

//to iterate key value use map entry

Map.Entry m1= (Map.Entry)itr.next();

//to get key and values we have to get by getKey and getValue

System.out.println("Map values "+" "+m1.getKey()+" "+m1.getValue());

}

}

}

package com.pioneercoders.mapimplementations;

import java.util.Collection;

import java.util.Hashtable;

import java.util.Set;

/\*

\* HashTable:-

1. It is a legacy class introduced in the 1.0 version.

2. Every method is synchronized hence only one thread is allow to access.

3. The performance of the application is low.

4. Null insertion is not possible if we are trying to insert null values we are getting NullPointerException.

\*/

public class HashTableEx {

public static void main(String[] args) {

Hashtable h=new Hashtable();

h.put("hyd",100);

h.put("banglore",200);

h.put("pune",300);

h.put("value", null);

/\*Exception in thread "main" java.lang.NullPointerException

at java.util.Hashtable.put(Hashtable.java:514)

at com.corejavaex.HashTableEx.main(HashTableEx.java:21)

\*/

System.out.println(h);

System.out.println(h.contains(300));//true

System.out.println(h.containsValue(200));//false

Collection c=h.values();

System.out.println(c);

Set c1=h.keySet();

System.out.println(c1);

}

}

package com.pioneercoders.mapimplementations;

import java.util.Iterator;

import java.util.LinkedHashMap;

import java.util.Map;

import java.util.Set;

/\*

\* introduced in 1.4 version

2) Heterogeneous data allowed.

3) Underlying data Structure is HashTable & linkedlist.

4) Duplicate keys are not allowed but values can be duplicated.

5) Insertion order is preserved.

6) Null is allowed for key(only once)and allows for values any number of times.

7) Every method is non-synchronized so multiple Threads are operate at a time hence permanence is high.

8) cursor :- Iterator

\*/

public class LinkedHashMapEx {

public static void main(String[] args) {

//creates LinkedList object with generic version

LinkedHashMap<Emp,Student> h = new LinkedHashMap<Emp,Student>();

h.put(new Emp(111,"ratan"), new Student(1,"budha"));

h.put(new Emp(222,"anu"), new Student(2,"ashok"));

//get Set interface before getting Iterator object

Set s = h.entrySet();

//creates iterator Object

Iterator itr = s.iterator();

while (itr.hasNext())

{//holding Entry by using Entry interface

Map.Entry m = (Map.Entry)itr.next();

Emp e = (Emp)m.getKey(); //getting Emp object

System.out.println(e.ename+"--"+e.eid);

Student ss = (Student)m.getValue(); //getting Student object

System.out.println(ss.sname+"--"+ss.sid);

}

}

}

package com.pioneercoders.mapimplementations;

public class Student {

//instance variables

int sid;

String sname;

Student(int sid,String sname)//local variables

{ this.sname=sname; this.sid=sid;

}

}

package com.pioneercoders.Setimplementations;

import java.util.HashSet;

public class HashSetExample {

public static void main(String[] args) {

//implementation class of Set interface

//Similar to list but it doesnt allows duplicates if u add also it doesnt raise any exception

//it doesnt add to the hashset

//it doesnt follows insertion order also

HashSet sett= new HashSet();

sett.add("sri");

sett.add("vamsi");

sett.add("vamsi");

sett.add(true);

sett.add(null);

System.out.println(sett);

}

}

package com.pioneercoders.Setimplementations;

import java.util.LinkedHashSet;

public class LinkedHashSetEx {

public static void main(String[] args) {

//similar to Hashset but it preserves insertion order

LinkedHashSet li= new LinkedHashSet();

li.add("srii");

li.add("anand");

li.add("mani");

li.add(null);

li.add("rajesh");

li.add("srii");

System.out.println(li);

}

}

package com.pioneercoders.Setimplementations;

import java.util.SortedSet;

import java.util.TreeSet;

public class SortedSetEx {

/\*

\* SortedSet methods

\* public E first(); it print first element

public E last(); it print last element

public E lower(E); it print lower object of specified object

public E higher(E); it print higher object of specified object

public java/util/SortedSet<E> subSet(E, E); it print subset

public java/util/SortedSet<E> headSet(E); it print specified object below

public java/util/SortedSet<E> tailSet(E); it print specified objects highest values and current element value

public E pollFirst(); it print and remove first

public E pollLast(); it print and remove last.

\*/

public static void main(String[] args) {

//creating TreeSet object

TreeSet t=new TreeSet();

//adding object in TreeSet

t.add(50); t.add(20); t.add(40);

t.add(10); t.add(30);

t.add(60);

t.add(70);

System.out.println(t);

//SortedSet added elements in sorting order by default it follows ascending order

//by using sortedset methods we can only some elements

SortedSet sett= t.headSet(60);//returns elements below 60

System.out.println(sett);

SortedSet sett1= t.tailSet(20);//it returns current element and return highest values

System.out.println(sett1);

SortedSet s3=t.subSet(20,50);//it print between values including lowest value also

System.out.println(s3); //[20,30,40]

System.out.println("last element="+t.last());//returns last element

System.out.println("first element="+t.first());//returns first element

System.out.println("lower element="+t.lower(50));//print

System.out.println("higher element="+t.higher(20));

System.out.println("print & remove first element="+t.pollFirst());//print 10 and remove that element

System.out.println("print & remove last element="+t.pollLast());//print 70 and removes that element

System.out.println("final elements="+t);//total elements

System.out.println("TreeSet size="+t.size());//it returns total elements size

System.out.println("TreeSet size="+t.remove(10));//it returns false because 10 is already deleted

System.out.println("TreeSet size="+t.remove(30));//it returns true and deletes value 30

System.out.println("Elements remaining"+t);

}

}

package com.pioneercoders.Setimplementations;

import java.util.Iterator;

import java.util.TreeSet;

public class TreeSetEx {

/\*

\* TreeSet:-

1. The underlying data Structure is BalencedTree.

2. Insertion order is not preserved it is based some sorting order.

3. Heterogeneous data is not allowed.

4. Duplicate objects are not allowed

5. Null insertion is possible only once.

\*/

public static void main(String[] args) {

// creating a TreeSet Object

TreeSet <Integer>treeadd = new TreeSet<Integer>();

// adding object in the tree set

treeadd.add(10);

treeadd.add(30);

treeadd.add(70);

treeadd.add(20);

// create iterator Object

Iterator iterator;

iterator = treeadd.iterator();

// displaying the Tree set data

System.out.println("Tree set data in ascending order: ");

while (iterator.hasNext()){

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

}

}

}

package com.pioneercoders;

import java.util.ArrayList;

import java.util.List;

public class GenericExample {

/\* Before going to discuss about Generics lets see some differences between array and collections to avoid drawbacks in collections we are using generics

\*

\* Arrays Collections

\* 1.It will provides type safety 1.It doesn't provide type safety it allows

\* it ensures that it allows only particular data any values like int,float,String values

\* type of values (if our requirement is to store in Single Collection

\* String values we will create that's y it doesnt provide type safety

\* String array and add values by mistake if

\* we add other than String values we will

\* get compiletime error

\*

\* 2. While Retriving values from array 2.While Retreiving values from String

\* it is not required to perform any type casting it is required to type cast otherwise we will get compiletimeerror because

\* in collection it will be having different type of values

\* -------------------------------------------------------------------------------------

\*

\* The Above Problems in Collection that will lead to go for Generics in Collections because type casting is bigger headache in normal collection(non-generic)

\*

\* Note: Generic main Advantage is ensures typesafety,and avoids explicit typecasting while retreiving

\*

\* The Java Generics programming is introduced in J2SE 5

\*

\* Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.

Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

1) Type-safety : We can hold only a single type of objects in generics. It doesn’t allow to store other objects.

2) Type casting is not required: There is no need to typecast the object.

Before Generics, we need to type cast the object while retrieving

List list = new ArrayList();

list.add("hello");

String s = (String) list.get(0);//typecasting

After Generics, we don't need to typecast the object while retrieving

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

list.add("hello");

String s = list.get(0);

3) Compile-Time Checking: It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

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

list.add("hello");

list.add(32);//Compile Time Error

\*

\*

\*/

public static void main(String[] args) {

/\*

\*

\*

\* Simply one point to remember about generics

\*

\* 1.To Ensure TypeSafety

\* 2.To Avoid Explicit Typecasting While Retrieving

\*

\*

\*

\*/

//simple example without generic

List list = new ArrayList(); //in this it allows any type of data we cannot ensure type safety

list.add("hello");

list.add(120);

String s = (String) list.get(0);//typecasting is required to get String values

System.out.println(s);

Integer i = (Integer)list.get(1);//typecasting is required to get String values

System.out.println(i);

/\*

\* Syntax to use generic collection

ClassOrInterface<Type>

Example to use Generics in java

ArrayList<String>

If we create with generic version like

ArrayList<String> li= new ArrayList<String>();

it creates the ArrayList of generic version which allows only String values if we enter any other values we will get ComplileTimeError

\* if we retrieve data we cannot typecast it because all are String values

\*/

//simple example with Generics, we can assure typesafety and we don't need to typecast the object while retrieving

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

li.add("my name is khan");

li.add("Singh is king");

li.add("Chak De India");

String str= li.get(0);//if we use generic version of Collection don't need to typecast while retrieving

String str1= li.get(1);//if we use generic version of Collection don't need to typecast while retrieving

String str2= li.get(2);//if we use generic version of Collection don't need to typecast while retrieving

System.out.println("Values are"+" "+str+" "+str1+" "+str2);

/\*

\* if u want to check whether generic version of collection ensures typesafety

\*

\* remove the comment on li.add(123);

\* and run the java application ,u will get the error

\*

\* Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The method add(int, String) in the type List<String> is not applicable for the arguments (int)

at com.pioneercoders.GenericExample.main(GenericExample.java:128)

\*/

// li.add(123);//if we add other String elements like int ,float we will get following error

/\*

\* Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The method add(int, String) in the type List<String> is not applicable for the arguments (int)

at com.pioneercoders.GenericExample.main(GenericExample.java:128)

\*/

}

}