**Polymorphism:**

* Poly means “many” and morphism means “forms”
* Same action in different way or one person with different behavior

**Inheritance:**

* Acquiring features from superclass (or) base class (or) parent class to sub class (child class) or (derived class)
* It is achieved by **“extends”** keyword
* It provides code reusability

**Ex:**

class Doctor {

void Doctor\_Details() {

System.out.println("Doctor Details...");

}

}

class Surgeon extends Doctor {

void Surgeon\_Details() {

System.out.println("Surgen Detail...");

}

}

public class Hospital {

public static void main(String args[]) {

Surgeon s = new Surgeon();

s.Doctor\_Details();

s.Surgeon\_Details();

}

}

## Super Keyword

The super keyword is similar to "this" keyword.

The keyword super can be used to access any data member or methods of the parent class.

Super keyword can be used at variable, method and constructor level.

**Syntax:**

super.<method-name>();

**Difference between string, String buffer and String builder**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **String** | **String buffer** | **String Builder** |
| 1. | Immutable | Mutable | Mutable |
| 2 | Thread safe | Thread safe | No thread safe |
| 3 | Stored in **“constant string pool”** | Stored in **“Heap”** | Stored in **“Heap”** |
| 4 | Performance is fast | Performance is very slow compare to string builder(beccz multi threading use) | Fast (jvm optimizations) |
| 5 | String str=”Hello”  Str=”Hi” | Stringbuffer s=new stringbuffer(“Hello”)  S=new stringbuffer(“Hi”) | Stringbuilder s=new lderstringbui(“Hello”)  S=new stringbuilder(“Hi”) |

**Difference between Final,Finally and finalize**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Final** | **Finally** | **Finalize** |
| 1. | Is an access modifier | is a block | Is a method of an object class |
| 2 | It is constant cant change  cant be overridden,  cant be extended | Associated with try catch block | Is a method applicable to object  Used for garbage collections |
| 3 | Final int a=5; | Try{  }  Catch{  }  Finally{  } | Protected void finalize throws throwable{  }  Public void finalize(){  } |
| 4 | Final class<clasname>  {  } |  |  |
| 5 | Final void rr()  {  } |  |  |

**Difference between Method overloading and method overriding**

|  |  |  |
| --- | --- | --- |
| **S. No** | **Method overloading** | **Method overriding** |
| 1. | Same name different no of arguments | Same name same no of arguments |
| 2 | Compile time polymorphism | Run time polymorphism |
| 3 | Return type may or may not be same | Return type should be same |
| 4 | Scope of overloading is within the class | Scope of overriding is base class and derived class |
| 5 | Private, static and final methods can be overloaded | Private, static and final methods cant be overloaded |
| 6 | Can be done at both static and non static methods | Can be done only at non static methods |

**Constructors:**

* Used to create and initialize the objects
* Is a special method in OOPlanguage
* Constructor name and class name must be same
* No return types
* Constructors cannot be abstract or static or final.
* Constructor can be overloaded but cannot be overridden

// Create a MyClass class  
public class MyClass {  
  int x;  // Create a class attribute  
  
  // Create a **class constructor** for the MyClass class  
  public MyClass() {  
    x = 5;  // Set the initial value for the class attribute x  
  }  
  
  public static void main(String[] args) {  
    MyClass myObj = new MyClass(); // Create an object of class MyClass (This will **call the constructor**)  
    System.out.println(myObj.x); // Print the value of x  
  }  
}  
  
// Outputs 5

**Types:**

1. Default constructor

2. Parameterized constructor

3. No Arg constructor

**1. Default constructor**

Constructor without parameter also known as zero constructor

Ex:

Class Test//class name

{

Void m1(){

Sopln(“m1 method”);

}

Test()//Default constructor

{

}

PSVM()

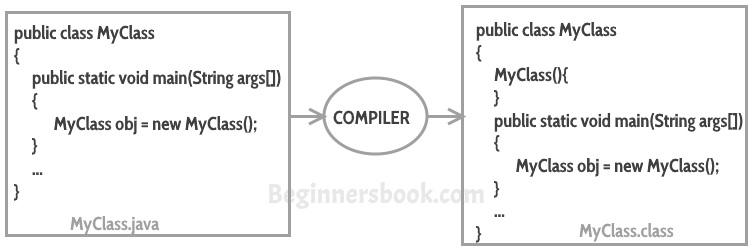
{

Test t=new Test();//t=object;Test()-constructor)

t.m1();

}

}



**2. User defined constructor or parameterized constructor**

Have parameters is known as parameterized constructor.

public class Employee {

int empId;

String empName;

//parameterized constructor with two parameters

Employee(int id, String name)//constructor with 2 parameters

{

this.empId = id;

this.empName = name;

}

void info(){

System.out.println("Id: "+empId+" Name: "+empName);

}

public static void main(String args[]){

Employee obj1 = new Employee(10245,"Chaitanya");

Employee obj2 = new Employee(92232,"Negan");

obj1.info();

obj2.info();

}

}

**Output:**

Id: 10245 Name: Chaitanya

Id: 92232 Name: Negan

**3. No-Arg Constructor** - a constructor that does not accept any arguments

class Demo

{

public Demo()

{

System.out.println("This is a no argument constructor");

}

public static void main(String args[]) {

new Demo();

}

}

Output:  
This is a no argument constructor

**What is mean by Exception?**

**Ans:** An Exception is a problem that can occur during the normal flow of an execution.

**What are the Advantages of Exception handling?**

**Ans: Given below are the advantages:**

* The normal flow of the execution won’t be terminated if exception got handled
* We can identify the problem by using catch declaration

Throwable

Errors Exception

Checked Unchecked

**Syntax:**

try{

statement 1;

statement 2;

}

Catch(Exception e){

s.o.pln(“Error in Try block”);

}

Finally{

s.o.pln(“final message”);//it will always run whether your code contain exception or not

}

**Throw:**

* Is a keyword in java which is used to throw an exception manually
* Using throw keyword can throw an exception from any method or block but it must be **“java.lang.Throwable”** class or sub-class

Ex:

Class test{

Private int sal =0;

Private int age=0;

Public int getsalary(){

If(salary<0){

throw new runtimeException(“salary cant lessthan zero”);

Return salary;

}

Publilc int getAge(){

If(age<0)

{

throw new runtimeException(“Age cant be -ve”);

return age;

}

}

**Throws:**

* Is a keyword in java
* Used in the signature of method to indicate that this method might throw one of the listed type exception

Ex:

Class test{

Private int sal =0;

Private int age=0;

Public int getsalary() throws RuntimeException{

return salary;

}

Publilc int getAge() throws Throwable {

If(age<0)

{

throw new runtimeException(“Age cant be -ve”);

return age;

}

Public int getMaxsal() throws IOException,,NumberformatException{

return 0;

}

}

**Throwable**

* Is a superclass of all exceptions and errors
* Is a interface not a class

Ex:

Public intgetAge() throws Throwable{

If(age>0)

Throw new RuntimeException(“Age cant be –ve”);

return age;

}

**Difference between Abstract and Interface**

|  |  |  |
| --- | --- | --- |
| **S. No** | **Abstract** | **Interface** |
| 1. | It have abstract and non abstract methods | It have only abstract methods |
| 2 | Doesn’t support multiple inheritance | support multiple inheritance |
| 3 | Used to declare the keyword as “Abstract” | Used to declare the keyword as “Interface” |
| 4 | It have final ,non-final, static and non static variable | It have only final and static variable |
| 5 | It have static methods and main method and constructor | It cant have static methods and main method and constructor |
| 6 | It have access modifier as public, private, protected ,static | Interface are implicitly public and abstract cant we any other access modifier |
| 7 | Subclass extend only one Abstract class | It can implement multiple interface |
| 6 | Can extend other class and implement interface | Can only extend other interface |
| 7 | **Abstract class:**  Abstract class A{  Data memers  }  Functions  {  }  **Abstract method:**  Class B extends Class A{  Method(){  }  } |  |

**Abstract:**

* Hiding the implementation details from the user and show only essential features

**Example:** Suppose you driving a car and you using all the functions of a car but you don't know how internally this function works

**Abstract class**

A class that is declared with abstract

It have abstract and non abstract methods(method with body)

**Ex:** abstract class A()

**Abstract Method:**

Is declared as abstract doesnot have any implementation

**Ex**: abstract void printstatus()

**Interface:**

* An interface is just like Java Class, but it **only has static constants and abstract method**.
* Java uses Interface **to implement multiple inheritance**.
* All methods in an interface are **implicitly public and abstract**.
* interfaces **can't have constructors**.
* it is used **to achieve fully abstraction** and it is a collection of abstract methods.
* Interface is a blueprint the can be used to call implement a class
* It doesnot have concrete methods
* All the methods of an interface are abstract and methods

**Syntax for Declaring Interface**

interface {

//methods

}

**What is Encapsulation in Java?**

Encapsulation is a mechanism of binding code and data together in a single unit. Let’s take an example of Capsule. Different powdered or liquid medicines are encapsulated inside a capsule.

**Purpose of Encapsulation:**

* Protects the code from others.
* Code maintainability.

**Difference between HashMap and HashTable can be seen below:**

| **HashMap** | **HashTable** |
| --- | --- |
| Methods are not synchronized | Key methods are synchronized |
| Not thread safety | Thread safety |
| Iterator is used to iterate the values | Enumerator is used to iterate the values |
| Allows one null key and multiple null values | Doesn’t allow anything that is null |
| Performance is high than HashTable | Performance is slow |

**22) Difference between HashSet and TreeSet.**

| **HashSet** | **TreeSet** |
| --- | --- |
| Inserted elements are in random order | Maintains the elements in the sorted order |
| Can able to store null objects | Couldn’t store null objects |
| Performance is fast | Performance is slow |

**Substring:**

public class StringMethodDemo

{

  public static void main(String args[])

  {

   String str1 = "ABCDEFGHI";                  // index numbering starts from 0

   String ss1 = str1.substring(3);          // returns DEFGHI

   String ss2 = str1.substring(3, 7);          // returns DEFG

   System.out.println("ss1:  " + ss1);

   System.out.println("ss2:  " + ss2);

                                                                               // other way to use

   System.out.println("abcdefgh".substring(4));// prints efgh

  }

}

charAt(int index) extracts a single character from a string.

Reverse a number without using reverse:

class ReverseNumberDemo

{

   public static void main(String args[])

   {

      int num=123456789;

      int reversenum =0;

      while( num != 0 )

      {

          reversenum = reversenum \* 10;

          reversenum = reversenum + num%10;

          num = num/10;

      }

      System.out.println("Reverse of specified number is: "+reversenum);

   }

}

Output:

Reverse of specified number is: 987654321

default: The scope of default access modifier is limited to the package only. If we do not mention any access modifier, then it acts like a default access modifier.

private: The scope of private access modifier is only within the classes.

Note: Class or Interface cannot be declared as private

protected: The scope of protected access modifier is within a package and also outside the package through inheritance only.

Note: Class cannot be declared as protected

public: The scope of public access modifier is everywhere. It has no restrictions. Data members, methods and classes that declared public can be accessed from anywhere.

See some sample programs of access modifiers

**What are the different Access modifiers available in Java?**

**Private:** If the class or method or variable is declared as private it’s scope will be only with in the class  
**public:** If the class or method or variable is declared as public it can be accessed anywhere within the class, package and outside the package  
**protected:** If the method or variable is declared as protected it can be accessed only within a packages where it is declared

**Difference between Array List and Vector in java?**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Array List** | **Vector** |
| 1 | ArrayList is a legacy class in java collection | Vector class is a legacy class in java collection |
| 2 | ArrayList class is not synchronized. | Vector class is synchronized class. |
| 3 | ArrayList class increases its size by 50% of the array size. | Vector class increases its size by doubling the array size. |

### **Difference between HashMap and HashSet?**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Hash Map** | **Hash Set** |
| 1 | HashMap class implements Map interface. | HashSet class implements Set interface. |
| 2 | HashMap class contains elements in the key-value combination.. | HashSet class contains only single value, there is no key-value combination. |

**Difference between Array List and Linked List in java?**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Array List** | **Linked List** |
| 1 | ArrayList uses a dynamic array to store the elements | LinkedList uses a doubly linked list to store data. |
| 2 | ArrayList is good for the store and fetch data. | LinkedList is good for manipulation of the data. |

### **Difference between HashMap and HashTable?**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Hash Map** | **Hash Table** |
| 1 | HashMap class is not synchronized. | Hashtable class is synchronized. |
| 2 | HashMap class allows one null key and multiple null values. | Hashtable class doesn't allow any null key and null values. |

### **Difference between List and Set?**

|  |  |  |
| --- | --- | --- |
| **S.No** | **List** | **Set** |
| 1 | List interface contains dupicate elements. | Set interface does not contains duplicate elements. |
| 2 | ArrayList, LinkedList, and Vector class implement List interface. | HashSet, LinkedHashSet, and TreeSet implements Set interface. |

**Fibonacci Series**

package softwareTestingMaterial;

public class FibonacciSeries {

public static void main(String args[]) {

int a = 0, b = 1, c, i, count = 10;

// To print 0 and 1

System.out.print(a + " " + b);

// loop starts from 2. We have already printed 0 and 1 in the previous step

for (i = 2; i < count; i++) {

c = a + b;

System.out.print(" " + c);

a = b;

b = c;

}

}

}

**Program to Check Palindrome using while loop**

public class Palindrome {

public static void main(String[] args) {

int num = 121, reversedInteger = 0, remainder, originalInteger;

originalInteger = num;

// reversed integer is stored in variable

while( num != 0 )

{

remainder = num % 10;

reversedInteger = reversedInteger \* 10 + remainder;

num /= 10;

}

// palindrome if orignalInteger and reversedInteger are equal

if (originalInteger == reversedInteger)

System.out.println(originalInteger + " is a palindrome.");

else

System.out.println(originalInteger + " is not a palindrome.");

}

}

When you run the program, the output will be:

121 is a palindrome number.

**Program to Check Alphabet using if else**

public class Alphabet {

public static void main(String[] args) {

char c = '\*';

if( (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

System.out.println(c + " is an alphabet.");

else

System.out.println(c + " is not an alphabet.");

}

}

When you run the program, the output will be:

\* is not an alphabet.

## Check whether an alphabet is vowel or consonant using if..else statement

public class VowelConsonant {

public static void main(String[] args) {

char ch = 'i';

if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' )

System.out.println(ch + " is vowel");

else

System.out.println(ch + " is consonant");

}

}

When you run the program, the output will be:

i is vowel

### Program to print half pyramid using

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**Source code**

public class Pattern {

public static void main(String[] args) {

int rows = 5;

for(int i = 1; i <= rows; ++i) {

for(int j = 1; j <= i; ++j) {

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

}

System.out.println();

}

}

}

## Get Current date and time in default format

import java.time.LocalDateTime;

public class CurrentDateTime {

public static void main(String[] args) {

LocalDateTime current = LocalDateTime.now();

System.out.println("Current Date and Time is: " + current);

}

}

When you run the program, the output will be:

Current Date and Time is: 2017-08-02T11:25:44.973

Java Program Example - Check Prime or Not \*/

import java.util.Scanner;

public class JavaProgram

{

public static void main(String args[])

{

int num, i, count=0;

Scanner scan = new Scanner(System.in);

System.out.print("Enter a Number : ");

num = scan.nextInt();

for(i=2; i<num; i++)

{

if(num%i == 0)

{

count++;

break;

}

}

if(count == 0)

{

System.out.print("This is a Prime Number");

}

else

{

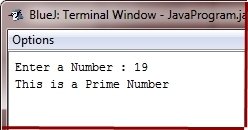
System.out.print("This is not a Prime Number");

}

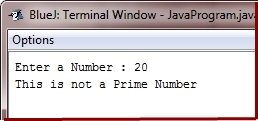
}

}

When the above Java Program is compile and executed, it will produce the following output. Above Java Programming Example Output (for prime number):



Above Java Programming Example Output (for not prime number):



**To check palindrome or not**

import java.util.Scanner;

public class JavaProgram

{

public static void main(String args[])

{

int num, rem, orig, rev=0;

Scanner scan = new Scanner(System.in);

System.out.print("Enter a Number : ");

num = scan.nextInt();

orig = num;

while(num != 0)

{

rem = num%10;

rev = rev\*10 + rem;

num = num/10;

}

// check if the original number is equal to its reverse

if(rev==orig)

{

System.out.print("This is a Palindrome Number");

}

else

{

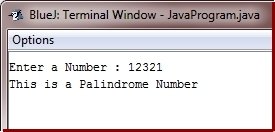
System.out.print("This is not a Palindrome Number");

}

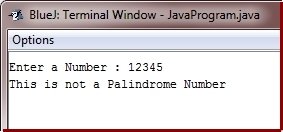
}

}

When the above Java Program is compile and executed, it will produce the following result. Above Java Programming Example Output (palindrome):



Above Java Programming Example Output (not palindrome):



**Palindrome string:**

import java.util.Scanner;

class ChkPalindrome

{

public static void main(String args[])

{

String str, rev = "";

Scanner sc = new Scanner(System.in);

System.out.println("Enter a string:");

str = sc.nextLine();

int length = str.length();

for ( int i = length - 1; i >= 0; i-- )

rev = rev + str.charAt(i);

if (str.equals(rev))

System.out.println(str+" is a palindrome");

else

System.out.println(str+" is not a palindrome");

}

}

**Output**

Enter a string:

radar

radar is a palindrome

### **Prime no or not**

public class PrimeNumbers {

public static void main(String[] args) {

int num = 20, count;

for (int i = 1; i & lt; = num; i++) {

count = 0;

for (int j = 2; j & lt; = i / 2; j++) {

if (i % j == 0) {

count++;

break;

}

}

if (count == 0) {

System.out.println(i);

}

}

}

Output:

1

2

3

5

7

11

13

17

19

# **Find Duplicate Characters in a String**

public class DuplStr {

public static void main(String argu[]) {

String str = "w3schools";

int cnt = 0;

char[] inp = str.toCharArray();

System.out.println("Duplicate Characters are:");

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

for (int j = i + 1; j < str.length(); j++) {

if (inp[i] == inp[j]) {

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

cnt++;

break;

}

}

}

}

}

Output:

Duplicate Characters are: s o

# **Even or Odd number**

import java.util.Scanner;

class CheckEvenOdd

{

public static void main(String args[])

{

int num;

System.out.println("Enter an Integer number:");

//The input provided by user is stored in num

Scanner input = new Scanner(System.in);

num = input.nextInt();

/\* If number is divisible by 2 then it's an even number

\* else odd number\*/

if ( num % 2 == 0 )

System.out.println("Entered number is even");

else

System.out.println("Entered number is odd");

}

}

Enter an Integer number:

78

Entered number is even

Enter an Integer number:

77

Entered number is odd