**UNIT-II**

**CLASS FUNDAMENTALS**

* Class—core of java
* Logical construct—define shape and nature of an object
* Class defines a new data type.
* This new type can be used to create objects of that type.
* ***Class is a template of an object and an object is an instance of a class.***
* Object—instance(interchangeably)
* Class is declared by using class keyword.

The general form of a class definition is:

class classname {

type instance-variabes1;//data or variables

type instance-variable2;

….

type instance-variableN;

type methodname1(parameter-list){

//body of the method

}

type methodname2(parameter-list){

//body of the method

}

}

* The data or variables, defined with in a class are called instance variables. The code is contained within methods.
* Methods and variables defined within a class are called members of the class.
* Instance of a class-object---object consists of same variables which are defined inside the class.
* Class can have any no of objects
* Methods:-- all the methods have general form as main()
* Most methods—not specified as public, static
* General form class does not specify main () method.
* Main()---starting point of your program
* Applets—no main () method at all

A SIMPLE CLASS

class Box {

double width;

double height;

double depth;

}

* Class defines a new type of data.
* The new data type is called Box.
* Class declaration creates—template
* It does not create an actual object.
* Create an object by using new

Box mybox= new Box();// object name mybox

* mybox will be an instance of Box. It will have “Physical” reality.
* Each time we create an instance of a class, we are creating an object that contains its own copy of each instance variable defined by the class. Thus, every Box object will contain its own copies of instance variables width, height, and depth.
* To access these variables, we have to use the dot (.) operator. The dot (.) operator links the name of the object with the name of an instance variable.
* For example, to assign the width variable of mybox the value 100, we would use the following statement.
* mybox.width=100;
* Dot (.) operator is used to access both the instance variables and the methods with in a object.
* Here is a complete program that uses the Box class:

/\* A program that uses the Box class.

Call this file BoxDemo.java

\*/

class Box {

double width;

double height;

double depth;

}

// This class declares an object of type Box.

class BoxDemo {

public static void main( String args[]) {

Box mybox=new Box();

double vol;

//assign values to mybox’s instance variables

mybox.width=10;

mybox.height=20;

mybox.depth=15;

//compute volume of box

vol= mybox.width\*mybox.height\*mybox.depth;

System.out.println(“Volume is” +vol);

}

}

* Compile—generates two .class files.1. Box and 2. BoxDemo
* Can put-each class in its own file, called Box.java and BoxDemo.java respectively.
* Output:

Volume is 3000.0

* Instance variables of one object have no effect on the instance variables of another.
* Let us declare two Box objects:

//Program declares two Box objects:

class Box {

double width;

double height;

double depth;

}

class BoxDemo2 {

public static void main( String args[]) {

Box mybox1=new Box();//object1=mybox1

Box mybox2=new Box(); //object2=mybox2

Double vol;

//assign values to mybox1’s instance variables

mybox1.width=10;

mybox1.height=20;

mybox1.depth=15;

//assign values to mybox2’s instance variables

mybox2.width=3;

mybox2.height=6;

mybox2.depth=9;

//compute volume of first box

vol= mybox1.width\*mybox1.height\*mybox1.depth;

System.out.println(“Volume of mybox1” +vol);

// compute volume of Second box

vol= mybox2.width\*mybox2.height\*mybox2.depth;

System.out.println(“Volume of mybox2” +vol);

}

}

Output produced by this program is shown here:

Volume of mybox1 3000.0

Volume of mybox2 162.0

* mybox1 data is completely separate from data contained in mybox2.

Declaring Objects:

* class creates a data type by using the type we can declare objects.
* Obtaining objects of a class is a two-step process.
* First, Declare variable of class type.
* This variable does not define an object. Instead, it is simply a variable that can refer to an object.
* Second, you must acquire an actual, physical copy of the object and assign it to that variable. We can do this by using new operator. The new operator dynamically allocates memory for an object allocated b new.
* This reference is then stored in the variable. Thus in java, all class objects must be dynamically allocated. Let’s look at the details of this procedure:
* Box mybox = new Box();
* This statement combines the two steps just described. It can be rewritten like this to show each step more clearly:

Box mybox; // declare reference to object

mybox=new Box(); //allocate a Box object

* A Closer Look at new:

General form:

class-var=new classname();

Statement Effect

Box mybox; mybox

|  |
| --- |
| width |
| height |
| depth |

mybox=new Box();

mybox

**Box object**

* Class-var is variable of the class type being created. The class name is the name of the class that is being instantiated. The class name followed by parenthesis specifies the constructor for the class.
* A constructor defines what occurs when an object of a class is created. Constructors are an important part of all classes and have many significant attributes. Most real –world classes explicitly define their own constructors within their class definition. If there is no explicit constructor –java will automatically supply a default constructor. (For Box we use default constructor).
* **Why you do not need to use new for integers or characters?**
  + **The answer is java’s primitive types are not implemented as objects. They are implemented as “normal” variables.**
  + **Objects have many features and attributes that require java to treat them differently than it treats the primitive types. By not applying the same overhead to the primitive types that applies to objects, Java can implement the primitive types more efficiently.**
* new-allocates memory for an object during run time. The advantage of this approach is that your program can create as many or as few objects as it needs during the execution of your program.
* Insufficient memory—throws a run time exception.

**Assigning Object Reference Variables:**

* Object reference variables act differently when assignment takes place.
* For example:

Box b1=new Box();

Box b2=b1;

* Which means b1 and b2 refer to the same object.
* The assignment of b1 and b2 did not allocate any memory or copy any part of the original object. It simply makes b2 refer to the same object as does b1. Thus any changes made to the object through b2 will affect the object to which b1 is referring, since they are the same object.
* The situation is depicted here:

|  |
| --- |
| Width |
| Height |
| Depth |

**b1**

**b2 Box object**

* Although b1 and b2 both refer to the same object, they are not linked in any other way. For example, a subsequent assignment to b1 will simple unhook b1 from the original object without affecting the object or affecting b2.

Box b1=new Box();

Box b2=b1;

// …….

b1=null;

Here , b1 has been set to null, but b2 still points to the original object.

NOTE: When you assign one object reference variable to another reference variable, you are not creating a copy of the object, you are only making a copy of the reference.

INTRODUCING METHODS:

* Classes usually consists of two things: instance variables and methods.
* The general form of a method:

type name(parameter-list)

{

//body of method

}

Here type specifies the type of data returned by the method. This can be any valid type, including the class type that we create.

* If the method does not return any value, its return type must be void.
* The name of the method is specified by name. This can be any legal identifier other than those already used by other items within the current scope.
* The parameter list is a sequence of type and identifier pairs separated by commas. Parameters are essentially variables that receive the value of the arguments passed to the method when it is called.
* If the method has no parameters, then the parameter list will be empty.
* Methods that have a return type other than void , return a value to the calling routine using the following form of the return statement.

return value;

* Here, value is the value returned.

Adding a Method to the Box Class:

* Methods are used to access the instance variables defined by the class. In fact, methods define the interface to most classes. This allows the class implementor to hide the specific layout of internal data structures behind cleaner method abstractions.
* We can define methods to provide access to data and also define methods that are used internally by the class itself.

// this program includes a method inside the box class.

class Box {

double width;

double height;

double depth;

// Display volume of a box

void volume() {

System.out.print(“Volume is”);

System.out.println(width\*height\*depth);

}

}

class BoxDemo3 {

public static void main(String[] args)

{

Box mybox1=new Box();

Box mybox2=new Box();

//assign values to mybox1’s instance variables

mybox1.width=10;

mybox1.height=20;

mybox1.depth=15;

//assign different values to mybox2’s instance variables

mybox2.width=3;

mybox2.height=6;

` mybox2.depth=9;

//display volume of first box

mybox1.volume();

//display volume of second box

mybox2.volume();

}

}

Output: Volume is 3000.0

Volume is 162.0

* Inside the volume () method: The instance variables width, height and depth are referred directly without preceding them with an object name or the dot operator. When a method uses an instance variable that is defined by its class, it does so directly, without explicitly reference to an object and without use of the dot operator.

Returning a Value:

// now, volume () returns the volume of a box.

class Box {

double width;

double height;

double depth;

// compute and return volume

double volume() {

return width\*height\*depth;

}

}

class BoxDemo4 {

public static void main(String[] args)

{

Box mybox1=new Box();

Box mybox2=new Box();

Double vol;

//assign values to mybox1’s instance variables

mybox1.width=10;

mybox1.height=20;

mybox1.depth=15;

//assign different values to mybox2’s instance variables

mybox2.width=3;

mybox2.height=6;

` mybox2.depth=9;

//get volume of first box

vol=mybox1.volume();

System.out.println(“Volume is” +vol);

//get volume of second box

vol=mybox2.volume();

System.out.println(“volume is” +vol);

}

}

As you can see, when volume() is called, it is put on the right side of an assignment statement. On the left is a variable, in this case vol, that will receive the value returned by volume(). This after

vol=mybox1.volume();

executes, the value of mybox.volume() is 3000 and this value then is stored in vol.

There are two important things to understand about returning values.

* The type of data returned by a method must be compatible with the return type specified by the method. For example, if the return type of some method is Boolean, you could not return an integer.
* The variable receiving the value returned by a method(such as vol) must also be compatible with the return type specified for the method.
* The call to volume() in the above program could have been used directly in println :

System.out.println(“Volume is” +mybox1.volume());

**Adding a Method That Takes Parameters:**

* Some methods do not need parameters, most do.
* Parameters allow a method to be generalized. That is, a parameterized method can operate on a variety of data and/or be used in a number of slightly different situations.
* For example:

Here is a method that returns the square of the number 10:

int square ()

{

return 10\*10;

}

* Returns the value of 10 squared, its use is very limited. However, if you modify the method so that it takes a parameter , as shown next, then you can make square() much more useful.

int square (int i)

{

return i\*i;

}

* Now, square () will return the square of whatever value it is called with. That is square() is now a general-purpose method that can compute the square of any integer value, rather than just 10.

Here is an example:

int x,y;

x=square(5); //x equals 25

x=square(9); // x equals 81

y=2;

x=square(y); //x equals 4

* In first call to square(), the value 5 will be passed into parameter i. In the Second call,i will receive the value 9. The third invocation passes the value of y, which is 2 in this example. Therefore, the square() is able to return the square of whatever data it is passed.
* Parameter: A Parameter is a variable defined by a method that receives a value when the method is called.
* Ex: In square(), i is a parameter.
* Argument: The argument is a value that is passed to a method when it is invoked.
* For example: square(100) passes 100 as an argument. Inside square(), the parameter i receives that value.
* // This program uses a parameterized method.

Class Box {

double width;

double height;

double depth;

}

// compute and return volume

double volume () {

return width\*height\*depth;

}

//sets dimensions of box

void setDim (double w, double h, double d) {

width=w;

height=h;

depth=d;

}

}

Class BoxDemo5 {

Public static void main(String args[])

{

Box mybox1 = new Box();

Box mybox2= new Box();

//initialize each box

mybox1.setDim(10,20,15);

mybox2.setDim(3,6,9);

//get volume of first box

vol=mybox1.volume();

System.out.println(“Volume is” + vol);

// get volume of second box

Vol= mybox2.volume();

System.out.println(“Volume is ” +vol);

}

}

* setDim() method is used to set the dimensions of each box. For ex:

mybox1.setDim(10,20,15);

when this code gets executed , 10 is copied into parameter w, 20 is copied into h, and 15 is copied into d. Inside setDim() the values of w,h, and d are then assigned to width, height and depth, respectively.

**CONSTRUCTORS:**

* Java allows objects to initialize themselves when they are created, this automatic initialization is performed through the use of a constructor.
* A constructor initializes an object immediately upon creation. It has the same name as the class in which it resides and is syntactically similar to a method. Once defined, the constructor is automatically called when the object is created, before the new operator completes.
* Constructor has no return type, not even void. This is because the implicit return type of a class constructor is the class type itself.
* It is the constructor’s job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object immediately.
* Let us rework on the Box example so that the dimensions of a box are automatically initialized when an object is constructed. To do so, replace setDim() with a constructor.

// Here, box uses a constructor to initialize the dimensions of a box

class Box {

double width;

double height;

double depth;

// This is the constructor for Box.

Box () {

System.out.println(“constructing Box”);

width=10;

height=10;

depth=10;

}

// compute and return volume

double volume () {

return width\*height\*depth;

}

}

Class BoxDemo6 {

Public static void main(String args[])

{

// declare, allocate, and initialize Box objects

Box mybox1 = new Box(); // constructing Box

Box mybox2= new Box(); // constructing Box

double vol;

//get volume of first box

vol=mybox1.volume();

System.out.println(“Volume is” + vol);

// get volume of second box

vol= mybox2.volume();

System.out.println(“Volume is ” +vol);

}

}

When this program is run, it generates the following results:

Constructing Box

Constructing Box

Volume is 1000.0

Volume is 1000.0

* Here, both mybox1 and mybox2 were initialized by the Box () constructor when they were created. Since the constructor gives all boxes the same dimensions, 10 by 10 by 10, both mybox1 and mybox2 will have the same volume.
* Box mybox1= new Box ();
* new Box() is calling the Box() constructor. When you do not explicitly define a constructor for a class, then java creates a default constructor for the class.
* The default constructor automatically initializes all instance variables to their default values, which are zero, null, and false , for numeric types, reference types, and Boolean, respectively.

Parameterized Constructors:

* While the Box() constructor in the preceding example does initialize a Box object, it is not very useful-all boxes have the same dimensions.
* We need a way to construct Box objects of various dimensions. The solution is to add parameters to the constructor.
* /\* Here, Box uses a parameterized constructor to

Initialize the dimensions of a box.

\*/

class Box {

double width;

double height;

double depth;

// this is the constructor for box.

Box (double w, double h, double d)

{

width=w;

height=h;

depth=d;

}

// compute and return volume

double volume () {

return width\*height\*depth;

}

}

Class BoxDemo7 {

public static void main(String args[])

//declare, allocate, and initialize Box objects

Box mybox1= new Box(10,20,15);

Box mybox2=new Box(3,6,9);

double vol;

//get volume of first box

vol= mybox1.volume();

System.out.println(“Volume is” +vol);

//get volume of second box

vol=mybox2.volume();

System.out.println(“Volume is” +vol);

}

}

The output:

Volume is 3000.0

Volume is 162.0

**The this Keyword:**

Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines this keyword. this keyword can be used inside any method to refer to the current object.

That is, this is always a reference to the object on which the method was invoked. We can use this anywhere a reference to an object of the current class type is permitted.

Example:

Box(double w, double h, double d) {

this.width = w;

this.height = h;

this.depth = d;

}

**Instance Variable Hiding:**

* It is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes.
* We can have local variables, including formal parameters to methods, which overlap with the names of the class instance variables.
* When the local variable has the same name as an instance variable, the local variable hides the instance variable. This is why width, height, and depth were not used as the names of the parameters to the Box () constructor inside Box class.
* If they had been, then width for example, would have referred to the formal parameter, hiding the instance variable width.
* To avoid problem, we can use different names, there is another way.
* this lets you refer directly to the object, you can use it to resolve any namespace collisions that might occur between instance variables and local variables.

// Use this to resolve name-space collisions.

Box (double width, double height, double depth) {

this.width= width;

this.height=height;

this.depth-depth;

}

Garbage collection: Objects are dynamically allocated by using new operator. When no reference to that object exist, the object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed, there is no explicit way to destroy objects. This is done Garbage collector. This technique that accomplishes this is called garbage collection. This will occur during the execution of your program.

The finalize () Method: Sometimes an object will need to perform some action when it is destroyed.

* For example, if an object is holding some non-java resource such as a file handle or character font, then you might want to make sure these resources are freed before an object is destroyed.
* To handle such situations, Java provides a mechanism called finalization. By using finalization you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.
* To add a finalizer to a class, we simple define the finalize () method.
* The Java runtime calls that method whenever it is about to recycle an object of that class.
* Inside the finalize () method, you will specify those actions that must be performed before an object is destroyed.
* The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. Right before the asset is freed, the java run time calls the finalize () method on the object.

// general form of finalize() method

Protected void finalize()

{

// finalization code here

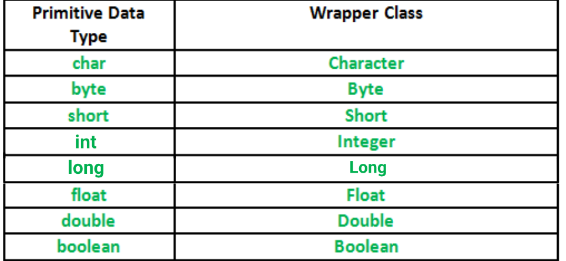
}

Protected keyword is a specifier that limits access to finalize().

finalize() is only called just prior to garbage collection.

**Wrapper Classes:**

* Wrapper classes provide a way to use primitive data types (int, boolean, etc..) as objects.
* Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data type. We can wrap a primitive value into a wrapper class object.
* Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.



* Use of Wrapper classes in Java
* Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.
* Change the value in Method: Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
* Serialization: We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
* Synchronization: Java synchronization works with objects in Multithreading.
* java.util package: The java.util package provides the utility classes to deal with objects.
* Collection Framework: Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

import java.util.ArrayList;

public class Main {

public static void main(String[] args) {

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

myNumbers.add(10);

myNumbers.add(15);

myNumbers.add(20);

myNumbers.add(25);

for (int i : myNumbers) {

System.out.println(i);

}

**}**

**}**

**Creating wrapper class:** To create a wrapper object, use the wrapper class instead of the primitive type. To get the value, you can just print the object:

public class Main {

public static void main(String[] args) {

Integer myInt = 5;

Double myDouble = 5.99;

Character myChar = 'A';

System.out.println(myInt);

System.out.println(myDouble);

System.out.println(myChar);

}

}

You can use certain methods to get information about the specific object.

For example, the following methods are used to get the value associated with the corresponding wrapper object: intValue(), byteValue(), shortValue(), longValue(), floatValue(), doubleValue(), charValue(), booleanValue().

This example will output the same result as the example above:

public class Main {

public static void main(String[] args) {

Integer myInt = 5;

Double myDouble = 5.99;

Character myChar = 'A';

System.out.println(myInt.intValue());

System.out.println(myDouble.doubleValue());

System.out.println(myChar.charValue());

}

}

Autoboxing

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects.

Wrapper class Example: Primitive to Wrapper

//Java program to convert primitive into objects

//Autoboxing example of int to Integer

public class WrapperExample1{

public static void main(String args[]){

//Converting int into Integer

int a=20;

Integer i=Integer.valueOf(a);//converting int into Integer explicitly

Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);

}}

Output:

20 20 20

Unboxing

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

Wrapper class Example: Wrapper to Primitive

//Java program to convert object into primitives

//Unboxing example of Integer to int

public class WrapperExample2{

public static void main(String args[]){

//Converting Integer to int

Integer a=new Integer(3);

int i=a.intValue();//converting Integer to int explicitly

int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);

}}

Output:

3 3 3

Java Wrapper classes Example

//Java Program to convert all primitives into its corresponding

//wrapper objects and vice-versa

public class WrapperExample3{

public static void main(String args[]){

byte b=10;

short s=20;

int i=30;

long l=40;

float f=50.0F;

double d=60.0D;

char c='a';

boolean b2=true;

//Autoboxing: Converting primitives into objects

Byte byteobj=b;

Short shortobj=s;

Integer intobj=i;

Long longobj=l;

Float floatobj=f;

Double doubleobj=d;

Character charobj=c;

Boolean boolobj=b2;

//Printing objects

System.out.println("---Printing object values---");

System.out.println("Byte object: "+byteobj);

System.out.println("Short object: "+shortobj);

System.out.println("Integer object: "+intobj);

System.out.println("Long object: "+longobj);

System.out.println("Float object: "+floatobj);

System.out.println("Double object: "+doubleobj);

System.out.println("Character object: "+charobj);

System.out.println("Boolean object: "+boolobj);

//Unboxing: Converting Objects to Primitives

byte bytevalue=byteobj;

short shortvalue=shortobj;

int intvalue=intobj;

long longvalue=longobj;

float floatvalue=floatobj;

double doublevalue=doubleobj;

char charvalue=charobj;

boolean boolvalue=boolobj;

//Printing primitives

System.out.println("---Printing primitive values---");

System.out.println("byte value: "+bytevalue);

System.out.println("short value: "+shortvalue);

System.out.println("int value: "+intvalue);

System.out.println("long value: "+longvalue);

System.out.println("float value: "+floatvalue);

System.out.println("double value: "+doublevalue);

System.out.println("char value: "+charvalue);

System.out.println("boolean value: "+boolvalue);

}}

**Command Line Arguments**

Sometimes you will want to pass information into a program when you run it. This is accomplished by passing command line arguments to main () method. A command-line argument is the information that directly follows the programs name on the command line when it is executed. To access the command line-arguments inside the java program is quite easy-they are stored as string in a String array passed to the args parameter of main().

The first command –line argument is stored at args[0], the second at args[1], and so on.

// Display all command-line arguments.

Class CommandLine {

Publis static void main(String args[]) {

for(int i=0, i<args.length;i++)

System.out.println(“args[“ + i + “ ]: ” +args[i]);

}

}

Try executing this program, as shown here:

java CommandLine this is a test 100 -1

we will get the following output:

args[0]:this

args[1]:is

args[2]:a

args[3]:test

args[4]:100

args[5]:-1

SCANNER CLASS:

Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings. It is the easiest way to read input in a Java program, though not very efficient if you want an input method for scenarios where time is a constraint like in competitive programming.

To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.

To read numerical values of a certain data type XYZ, the function to use is nextXYZ(). For example, to read a value of type short, we can use nextShort()

To read strings, we use nextLine().

To read a single character, we use next().charAt(0). next() function returns the next token/word in the input as a string and charAt(0) function returns the first character in that string.

import java.util.Scanner; // Import the Scanner class

class Main {

public static void main(String[] args) {

Scanner myObj = new Scanner(System.in); // Create a Scanner object

System.out.println("Enter username");

String userName = myObj.nextLine(); // Read user input

System.out.println("Username is: " + userName); // Output user input

}

}

// Java program to read data of various types using Scanner class.

import java.util.Scanner;

public class ScannerDemo1

{

public static void main(String[] args)

{

// Declare the object and initialize with

// predefined standard input object

Scanner sc = new Scanner(System.in);

// String input

String name = sc.nextLine();

// Character input

char gender = sc.next().charAt(0);

// Numerical data input

// byte, short and float can be read

// using similar-named functions.

int age = sc.nextInt();

long mobileNo = sc.nextLong();

double cgpa = sc.nextDouble();

// Print the values to check if the input was correctly obtained.

System.out.println("Name: "+name);

System.out.println("Gender: "+gender);

System.out.println("Age: "+age);

System.out.println("Mobile Number: "+mobileNo);

System.out.println("CGPA: "+cgpa);

}

}

Input :

Geek

F

40

9876543210

9.9

Output :

Name: Geek

Gender: F

Age: 40

Mobile Number: 9876543210

CGPA: 9.9

BufferedReader Class:

Java BufferedReader Class

Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader

class

.

Java BufferedReader class declaration

Let's see the declaration for Java.io.BufferedReader class:

public class BufferedReader extends Reader

Java BufferedReader class constructors

Constructor Description

BufferedReader(Reader rd) It is used to create a buffered character input stream that uses the default size for an input buffer.

BufferedReader(Reader rd, int size) It is used to create a buffered character input stream that uses the specified size for an input buffer.

Java BufferedReader class methods

Method Description

int read() It is used for reading a single character.

int read(char[] cbuf, int off, int len) It is used for reading characters into a portion of an array

.

boolean markSupported() It is used to test the input stream support for the mark and reset method.

String readLine() It is used for reading a line of text.

boolean ready() It is used to test whether the input stream is ready to be read.

long skip(long n) It is used for skipping the characters.

void reset() It repositions the stream

at a position the mark method was last called on this input stream.

void mark(int readAheadLimit) It is used for marking the present position in a stream.

void close() It closes the input stream and releases any of the system resources associated with the stream.

Java BufferedReader Example

In this example, we are reading the data from the text file testout.txt using Java BufferedReader class.

package com.javatpoint;

import java.io.\*;

public class BufferedReaderExample {

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

FileReader fr=new FileReader("D:\\testout.txt");

BufferedReader br=new BufferedReader(fr);

int i;

while((i=br.read())!=-1){

System.out.print((char)i);

}

br.close();

fr.close();

}

}

Here, we are assuming that you have following data in "testout.txt" file:

Welcome to javaTpoint.

Output:

Welcome to javaTpoint.

Reading data from console by InputStreamReader and BufferedReader

In this example, we are connecting the BufferedReader stream with the InputStreamReader

stream for reading the line by line data from the keyboard.

package com.javatpoint;

import java.io.\*;

public class BufferedReaderExample{

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

InputStreamReader r=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(r);

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

String name=br.readLine();

System.out.println("Welcome "+name);

}

}

Output:

Enter your name

Nakul Jain

Welcome Nakul Jain

Java Buffer reader class 1

Another example of reading data from console until user writes stop

In this example, we are reading and printing the data until the user prints stop.

package com.javatpoint;

import java.io.\*;

public class BufferedReaderExample{

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

InputStreamReader r=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(r);

String name="";

while(!name.equals("stop")){

System.out.println("Enter data: ");

name=br.readLine();

System.out.println("data is: "+name);

}

br.close();

r.close();

}

}

Output:

Enter data: Nakul

data is: Nakul

Enter data: 12

data is: 12

Enter data: stop

data is: stop