**Some other OOPS concepts:**

Math class: provides several methods to work on math claculations like min(),max(), avg(), etc….

Example:

**package** com.lokesh;

**public** **class** HelloWorld2{

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

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

**int** a =20;

**int** b=10;

System.***out***.println("Max number is "+Math.*max*(a, b));

}

}

Wrapper class: used to convert primitive into object and object into primitive

Uses –

Change the value in method – java supports only call by value, so if we pass primitive value it will not change original value. But if we can convert primitive value to object it will change original value

Serialization – convert object into streams to perform serialization. If we have primitive value, we can convert it in objects through wrapper classes

Synchronization – work with objects in multithreading

The eight class of java.lang package are knows as wrapper classes. Below are the list

|  |  |
| --- | --- |
| Primitive Type | Wrapper Class |
| boolean | Boolean |
| int | Character |
| byte | Byte |
| short | Short |
| integer | Integer |
| long | Long |
| float | Float |
| double | Double |

Autoboxing - automatic conversion from primitive to wrapper. For example byte to Byte, int to Int ..etc

**package** com.lokesh;

**public** **class** HelloWorld2{

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

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

**int** a =20;

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

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

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

}

}

Unboxing - automatic conversion from wrapper to primitive. For example Byte to byte, Int to int ..etc

**package** com.lokesh;

**public** **class** HelloWorld2{

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

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

Integer a=**new** ~~Integer~~(20);

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

**int** j =a; // autoboxing, compiler will write a.intValue() internally

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

}

}

Below is an Wrapper class example

**package** com.lokesh;

**public** **class** HelloWorld2{

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

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

**byte** a =10;

**short** b =20;

**int** c = 30;

//Autoboxing - primitive to objects

Byte d =a;

Short e =b;

Integer f =c;

System.***out***.println(d+" "+e+" "+f);

//Unboxing - objects to primitive

**byte** g=d;

**short** h=b;

**int** i=f;

System.***out***.println(g+" "+h+" "+i);

}

}

Below is a Custom Wrapper class example

**package** com.lokesh;

**public** **class** HelloWorld{

**int** id;

**public** HelloWorld(**int** id) {

**this**.id =id;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

@Override

**public** String toString() {

**return** Integer.*toString*(id);

}

}

**package** com.lokesh;

**public** **class** HelloWorld2{

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

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

HelloWorld hw = **new** HelloWorld(20);

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

}

}

Recursion: a process in which a method calls itself continuously

Below is an example for recursive call for infinite times

**package** com.lokesh;

**public** **class** HelloWorld{

**public** **void** method1() {

System.***out***.println("this is example for recursive infinite call");

method1();

}

}

**package** com.lokesh;

**public** **class** HelloWorld2{

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

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

HelloWorld hw = **new** HelloWorld();

hw.method1();

}

}

Below is an example for recursive call for finite times

**package** com.lokesh;

**public** **class** HelloWorld{

**public** **int** count=0;

**public** **void** method1() {

count++;

**if**(count <5) {

System.***out***.println("this is example for recursive finite call");

method1();

}

}

}

**package** com.lokesh;

**public** **class** HelloWorld2{

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

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

HelloWorld hw = **new** HelloWorld();

hw.method1();

}

}

Call by value: there is only call by value in java, not call by reference. If we call a method passing a value it is called pass by value. the Changes being done in called method is not affected in calling method

Below is an example of call by value where original value is not changed

**package** com.lokesh;

**public** **class** HelloWorld{

**public** **int** a=20;

**public** **void** method1(**int** a) {

a = a + 50; // changes will be in local variable only

}

}

**package** com.lokesh;

**public** **class** HelloWorld2{

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

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

HelloWorld hw = **new** HelloWorld();

System.***out***.println(hw.a);

hw.method1(30);

System.***out***.println(hw.a);

}

}

In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value. Below is an example

**package** com.lokesh;

**public** **class** HelloWorld{

**public** **int** a=20;

**public** **void** method1(HelloWorld hw1) {

hw1.a = hw1.a + 50; //changes will be in instance variable

}

}

**package** com.lokesh;

**public** **class** HelloWorld2{

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

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

HelloWorld hw2 = **new** HelloWorld();

System.***out***.println(hw2.a);

hw2.method1(hw2);

System.***out***.println(hw2.a);

}

}

strictfp keyword: ensures you will get same result on every platform if you perform operating in floating point variable

strictfp keyword can be applied on methods, classes and interfaces

strictfp class A{}

strictfp interface B{}

class A{

strictfp method1(){}

}

Java Regex (Regular expression): an API to define a pattern for searching strings.

Used for password and email validation

Pattern generally provides the data what to be searched for example “..j.” means the expectation is two words before j and one word after j

Matcher provides in which string it should search

Matches checks with pattern data with matcher data

Example:

**package** com.lokesh;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** HelloWorld2 {

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

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

//1st way

Pattern p =Pattern.*compile*(".a."); // here each dot means each word so there is one word before a and after a

Matcher m=p.matcher("ram");

**boolean** b =m.matches();

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

//2nd way

**boolean** b2 =Pattern.*compile*("..m").matcher("ram").matches();

System.***out***.println(b2);

//3rd way

**boolean** b3 =Pattern.*matches*("r..", "ram");

System.***out***.println(b3);

} s

}

Inner classes: class that is declared inside the class or interface

It can access all the members of the outer class, including private data members and methods

Syntax of Inner class:

lass Java\_Outer\_class{

//code

Class Java\_Inner\_class{

//code

}

}

Advantages – it can access all the members (data members and methods) of outer class including private

Non-static nested classes are called inner classes

Types

Member inner class

Anonymous inner class

Local inner class

Static nested class

Nested Interface

Member inner class: non-static class that is created inside a class outside a method. It can be declared with access modifiers like public, default, private and protected

Example:

**package** com.lokesh;

**import** java.io.IOException;

**public** **class** HelloWorld{

**private** **int** a =30;

**public** **class** HelloWorldsubclass{

**public** **void** method1() {

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

}

}

}

**package** com.lokesh;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**public** **class** HelloWorld2 {

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

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

HelloWorld hw=**new** HelloWorld();

HelloWorld.HelloWorldsubclass hwin=hw.**new** HelloWorldsubclass(); //instantiating inner class

hwin.method1();

}

}

Anonymous Inner class: an inner class without a name and for which only single object is created

In simple words a class that has no name is called anonymous inner class

Can be created in two ways – class (may be abstract or concrete), Interface

Example using class:

**package** com.lokesh;

**import** java.io.IOException;

**public** **abstract** **class** HelloWorld{

**public** **abstract** **void** method1();

}

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

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

//below part is anonymous inner class procedure where method implementation done inside the instantiation of object

HelloWorld hw=**new** HelloWorld() {

**public** **void** method1() {

System.***out***.println("this is method1");

}

};

hw.method1();

}

}

Example using interface:

**package** com.lokesh;

**public** **interface** HelloWorld1 {

**public** **void** method1();

}

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

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

//below part is anonymous inner class procedure where method implementation done inside the instantiation of object

HelloWorld1 hw=**new** HelloWorld1() {

**public** **void** method1() {

System.***out***.println("this is method1");

}

};

hw.method1();

}

}

Local inner class: A class that is created inside a method

Example: In below example after instantiating Helloworld class and called method1 from method1 HelloWorldsubclass has been instantiated and method2 has been called

**package** com.lokesh;

**import** java.io.IOException;

**public** **class** HelloWorld{

**private** **int** a=30;

**public** **void** method1() {

**class** HelloWorldsubclass{

**public** **void** method2() {

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

}

}

HelloWorldsubclass hwsub =**new** HelloWorldsubclass();

hwsub.method2();

}

}

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

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

HelloWorld hw = **new** HelloWorld();

hw.method1();

}

}

Static nested class: A static class created inside the class. It cannot access non-static data members and methods. It can be accessed by outer class name

It can access static data members and methods including private

Example:

**package** com.lokesh;

**import** java.io.IOException;

**public** **class** HelloWorld{

**static** **int** *a*=30;

**static** **class** HelloWorldsubclass{

**public** **void** method1() {

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

}

}

}

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

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

HelloWorld.HelloWorldsubclass hw=**new** HelloWorld.HelloWorldsubclass();

hw.method1();

}

}

Nested Interface: An interface declared within another interface or class.

Example of nested interface which is declared within interface

**package** com.lokesh;

**public** **interface** HelloWorld1 {

**public** **void** method1();

**interface** HelloWorldSubInterface{

**public** **void** method2();

}

}

**package** com.lokesh;

**public** **class** HelloWorld **implements** HelloWorld1.HelloWorldSubInterface{

**public** **void** method2() {

System.***out***.println("this is method of nested interface");

}

}

**package** com.lokesh;

**import** com.lokesh.HelloWorld1.HelloWorldSubInterface;

**public** **class** HelloWorld2 {

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

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

HelloWorldSubInterface hwinf =**new** HelloWorld();

hwinf.method2();

}

}

Example of nested interface which is declared within class

**package** com.lokesh;

**public** **class** HelloWorld{

**interface** HelloWorldinterface{

**public** **void** method1();

}

}

**package** com.lokesh;

**public** **class** HelloWorld2 **implements** HelloWorld.HelloWorldinterface {

**public** **void** method1() {

System.***out***.println("this is method of nester interface in class");

}

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

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

HelloWorld.HelloWorldinterface hwinf =**new** HelloWorld2();

hwinf.method1();

}

}

Java Reflection: is a process of examining and modifying the runtime behaviour of class at runtime.

java.lang and java.lang.reflect packages provide classes for java reflection

java.lang.class provides methods to get the metadata of class at runtime and provides methods to examine and change the runtime behaviour of class

There are three ways to get instance of Class class

forName() method

getClass() method

the.class syntax

forName() – used to load the class dynamically, returns the instance of Class class. It should be used if you know fully qualified name of class. This cannot be used for primitive datatypes

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

Class c =Class.*forName*("com.lokesh.HelloWorld2");

System.***out***.println(c.getName());

}

}

getClass() method – It returns the instance of Class class. It should be used if you know type. It can be used with primitives

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

**public** **void** printName(Object obj) {

Class c=obj.getClass();

System.***out***.println(c.getName());

}

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

HelloWorld1 hw =**new** HelloWorld1();

HelloWorld2 hw2 =**new** HelloWorld2();

hw2.printName(hw);

}

}

The .class syntax – If a type is available, but there is no instance, then it is possible to obtain a Class by appending “.class” to the name of type. It can be used for primitive data types also.

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

Class c=**boolean**.**class**;

System.***out***.println(c.getName());

Class c2=HelloWorld1.**class**;

System.***out***.println(c2.getName());

}

}

Java reflection should always be used with caution

Advantages – inspection of classes, methods ,interfaces and fields during runtime using reflection even without using their names during the compile time

Dis-advantages – using reflection one can break the principles of encapsulation. It is possible to access the private methods and fields of class using reflection. Thus reflection may leak important data to outside world.

Conclusion – it is generally advisable to avoid reflection. Whenever reflection is used the security of application is compromised

newInstance() method – the newInstance() method of Class class and Constructor is used to create a new instance of the class

The newInstance() method of Class class can invoke zero-argument constructor, whereas newInstance() method of Constructor class can invoke any number of arguments. So Constructor class is preferred over Class class

Example: Below is an example of creating an instance of class and invoking a method

**package** com.lokesh;

**public** **class** HelloWorld1 {

**public** **void** method1() {

System.***out***.println("this is method1");

}

}

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

Class c=Class.*forName*("com.lokesh.HelloWorld1");

HelloWorld1 hw1=(HelloWorld1) c.~~newInstance~~();

hw1.method1();

}

}

Javap tool: javap command disassembles a class file. The javap command displays information about the fields, constructors and methods in a class file.

Calling private methods and Constructors from another class – we can do it by changing runtime behaviour of the class

With the help of java.lang.Class and java.lang.reflect.Method we can call private method from another class

Example:

**package** com.lokesh;

**public** **class** HelloWorld1 {

**public** **void** method1() {

System.***out***.println("this is method1");

}

**public** **void** method2(**int** a) {

System.***out***.println("this is parameterized method");

System.***out***.println(a\*a\*a);

}

}

**package** com.lokesh;

**import** java.lang.reflect.Method;

**public** **class** HelloWorld2 {

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

Class c=Class.*forName*("com.lokesh.HelloWorld1");

Object obj =c.~~newInstance~~();

//invoking non-parameterized method

Method m1 =c.getDeclaredMethod("method1", **null**);

m1.setAccessible(**true**);

m1.invoke(obj, **null**);

//invoking parameterized method

Method m2=c.getDeclaredMethod("method2", **new** Class[] {**int**.**class**});

m2.setAccessible(**true**);

m2.invoke(obj, 2);

}

}

Example: Below is an example of invoking private non-parameterized and parameterized constructor

**package** com.lokesh;

**public** **class** HelloWorld1 {

**private** **int** a;

**private** **int** b;

**private** HelloWorld1() {

**this**.a=10;

**this**.b=20;

}

**private** HelloWorld1(**int** a, **int** b) {

**this**.a=a;

**this**.b=b;

}

**public** **int** getA() {

**return** a;

}

**public** **void** setA(**int** a) {

**this**.a = a;

}

**public** **int** getB() {

**return** b;

}

**public** **void** setB(**int** b) {

**this**.b = b;

}

}

**package** com.lokesh;

**import** java.lang.reflect.Constructor;

**import** java.lang.reflect.Method;

**public** **class** HelloWorld2 {

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

Class c=Class.*forName*("com.lokesh.HelloWorld1");

Constructor<HelloWorld1> hw1 =c.getDeclaredConstructor();

hw1.setAccessible(**true**);

Object obj1 =hw1.newInstance();

//invoking private non=parameterized private constructor

**if**(obj1 **instanceof** HelloWorld1) {

HelloWorld1 hw11 =(HelloWorld1) obj1;

System.***out***.println(hw11.getA()+" "+hw11.getB());

}

//setting values via setter methods

**if**(obj1 **instanceof** HelloWorld1) {

HelloWorld1 hw11 =(HelloWorld1) obj1;

hw11.setA(5);

hw11.setB(8);

System.***out***.println(hw11.getA()+" "+hw11.getB());

}

//invoking parameterized private constructor

Constructor<HelloWorld1> hw12=c.getDeclaredConstructor(**int**.**class**, **int**.**class**);

hw12.setAccessible(**true**);

Object obj2=hw12.newInstance(2,23);

**if**(obj2 **instanceof** HelloWorld1) {

HelloWorld1 hw22 =(HelloWorld1) obj2;

System.***out***.println(hw22.getA()+" "+hw22.getB());

}

}

}

Java Date and Time: java.time, java.util, java.sql and java.text packages contains classes for representing date and time

Using package java.util

Example: for date

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

//1st way

Date dt1=**new** Date();

System.***out***.println(dt1);

//2nd way

**long** t1=System.*currentTimeMillis*();

Date dt2 =**new** Date(t1);

System.***out***.println(dt2);

}

}

Example: Using Calendar method from java.util to get date

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

Calendar c1= Calendar.*getInstance*();

System.***out***.println(c1.getTime());

}

}

Timezone class – represents time zone offset, also figures out daylight savings.

Example:

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

//to get the list of time zones

String s1[] =TimeZone.*getAvailableIDs*();

**for**(**int** i=0;i<s1.length;i++) {

System.***out***.println(s1[i]);

}

//to get particular time offset of a timezone

TimeZone tz=TimeZone.*getTimeZone*("Asia/Kolkata");

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

}

}

Using Java.sql package

Example: To get date

**package** com.lokesh;

**import** java.sql.\*;

**public** **class** HelloWorld2 {

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

**long** t1=System.*currentTimeMillis*();

Date dt=**new** Date(t1);

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

}

}

Using java.time package

Example: To get date

**package** com.lokesh;

**import** java.time.\*;

**public** **class** HelloWorld2 {

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

LocalDate dt=LocalDate.*now*();

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

}

}

Example: To get time

**package** com.lokesh;

**import** java.time.\*;

**public** **class** HelloWorld2 {

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

LocalTime dt=LocalTime.*now*();

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

}

}

Java Conversion: Converting from one type to another type

Int to String - to conver String to int we use Integer.parseInt() and to convert String to Integer we use Integer.valueOf()

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

String s1 ="200";

//String to int

**int** i1 =Integer.*parseInt*(s1);

System.***out***.println(i1);

//String to Integer

Integer i2=Integer.*valueOf*(s1);

System.***out***.println(i2);

}

}

Example : Below is an example where you cannot convert String to int or Integer since String is not of numbers it throws NumberFormatException

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

String s1 ="ABCD";

//String to int

**int** i1 =Integer.*parseInt*(s1);

System.***out***.println(i1);

//String to Integer

Integer i2=Integer.*valueOf*(s1);

System.***out***.println(i2);

}

}

Int to String - we can convert using String.valueOf() and Integer.toString() and also String.format() methods

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**int** i=10;

//1st method

String s1 =String.*valueOf*(i);

//2nd method

String s2 =Integer.*toString*(i);

//3rd method

String s3=String.*format*("%d",i);

System.***out***.println(s1);

System.***out***.println(s2);

System.***out***.println(s3);

}

}

String to long – we convert using Long.parseLong()

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

String s1="3456896";

**long** l1 =Long.*parseLong*(s1);

System.***out***.println(l1);

}

}

long to String – we can convert using String.valueOf() and Long.toString() methods

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**long** l1=3456896L;

String s1 =String.*valueOf*(l1);

String s2=Long.*toString*(l1);

System.***out***.println(s1);

System.***out***.println(s2);

}

}

String to float – we can convert using Float.parseFloat()

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

String s1 ="22.3";

**float** f1 =Float.*parseFloat*(s1);

System.***out***.println(f1);

}

}

float to String – we can covet using String.valueOf() and Float.toString()

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**float** f1 =22.3f;

String s1 =String.*valueOf*(f1);

String s2=Float.*toString*(f1);

System.***out***.println(s1);

System.***out***.println(s2);

}

}

String to double – we can convert using Double.parseDouble()

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

String s1="23.6";

**double** d1=Double.*parseDouble*(s1);

System.***out***.println(d1);

}

}

double to String – we can convert using String.valueOf() and Double.toString()

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**double** d1=26.3;

String s1=String.*valueOf*(d1);

String s2=Double.*toString*(d1);

System.***out***.println(s1);

System.***out***.println(s2);

}

}

String to char – we can convert using charAt() method

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

String s1="hello";

**char** c1=s1.charAt(0); //returns h

**char** c2=s1.charAt(3); //returns l

System.***out***.println(c1);

System.***out***.println(c2);

}

}

char to String – we can convert using String.valueOf() and Character.toString() methods

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**char** c1='c';

String s1=String.*valueOf*(c1);

String s2=Character.*toString*(c1);

System.***out***.println(s1);

System.***out***.println(s2);

}

}

String to object – we can convert using assignment “=” operator

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

String s1="hello";

Object ob1=s1;

System.***out***.println(ob1);

}

}

Object to String – we can convert using toString() method

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

HelloWorld2 hw1=**new** HelloWorld2();

String s1=hw1.toString();

System.***out***.println(s1);

}

}

int to long – we can convert into long using assignment “=” operator

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**int** a=1000;

**long** b=a;

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

}

}

long to int – we can convert long to int using typecasting and intValue()

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**long** a=200;

**int** b=(**int**)a; //typecasting

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

Long c=**new** ~~Long~~(400);

**int** d=c.intValue();

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

}

}

int to double – we can convert using “=” operator

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**int** a=200;

**double** b=a;

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

}

}

double to int – we can convert using typecasting and inValue() methods

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**double** a=200.98d;

**int** b=(**int**)a; //typecasting converts to 200

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

Double d=**new** ~~Double~~(223.78);

**int** c=d.intValue();

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

}

}

char to int – we can convert using “=” operator (returns ASCII value) and Character.getNumericValue() and String.valueOf()

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**char** c1='1';

**int** a=c1; //returns ASCII value

**int** b=Character.*getNumericValue*(c1); //returns 1

**int** c=Integer.*parseInt*(String.*valueOf*(c1)); //returns 1

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

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

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

}

}

int to char – we can convert using typecasting

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

**int** a =65;

**char** c=(**char**)a; //returns ASCII character

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

}

}