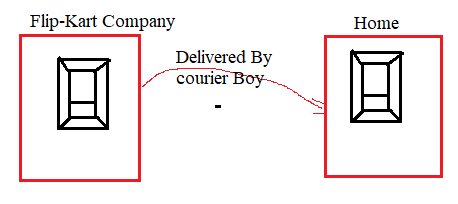
1. **Introduction**
2. **Wrapper classes.**
3. **Converting Primitive DT to Wrapper class objects.**
4. **Converting Wrapper class objects into Primitive DT.**
5. **Auto Boxing & Auto Un Boxing.**
6. **Wrapper class Casting.**
7. **Wrapper class objects Comparision.**

**INTRODUCTION**



The Flipkart company pack the mobile . It delivers pack to customer through courier boy. The customer first unpack it and take mobile . He starts using mobile.

Accoring to java Terminology

Mobile is primitive value.

Box is wrapper class.

Problem:-Collection is meant for storing multiple values or objects as one group with single name without type and size limitation. **The collection by default allows only objects.**

We can’t store primitive values in collection directly.

1. Programmer can develop custom wrapper class to solve above problem. Where as by developing custom wrapper classes, There are few disadvantages.
   1. Programmer has to write 8 wrapper classes to represent primitive values as object in every project . It is time waste further money waste.
   2. When programmer changed to new company, programer has to known about wrapper classes.

Solution:- The sun micro system overcome this problem by introducing wrapper classes.

**WRAPPER CLASSES**

**1.Wrapping**:- Wrapping is enclosing/covering somethind with paper.

As per Java, Wrapping is storing priitive value inside one class object for representing it as object.

**2. Wrapper classes:-** The class which meant for wrapping/converting primitive value to represent value it as object is called wrapper class.

We find a method parameter or return type as Object class type. To this method, we need to pass | return primitive type value as argument | return value then we must use wrapper class object.

Example:1

Class Example{

Static void m1(Object x){

}

}

Example:2

Class Example1{

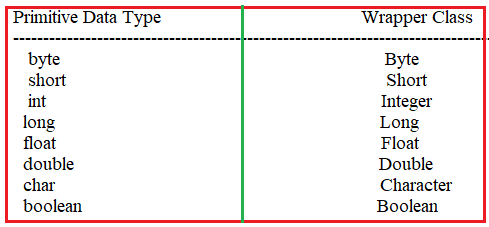
Static Object m2(){

Return ?

}

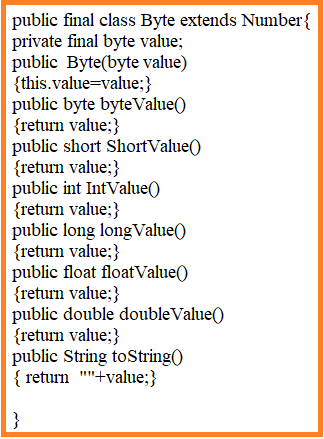
}

**3. Types of Wrapper classes:-** The table have primitve data type names and its corresponding wrapper class names.

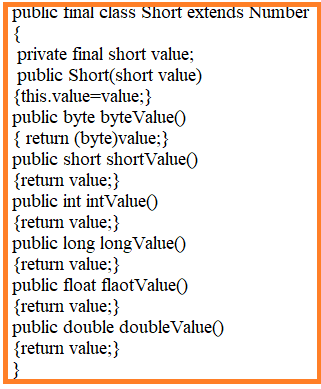


4. Wrapper classes Templates:

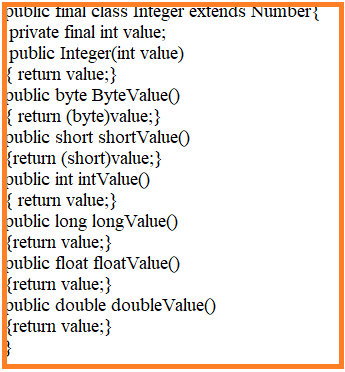
ClassName:Byte



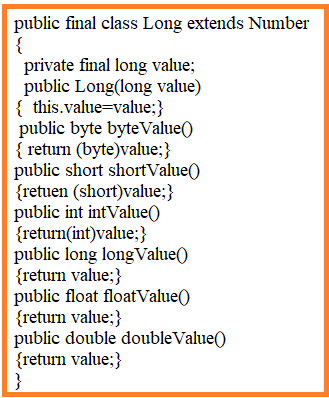
className: Short



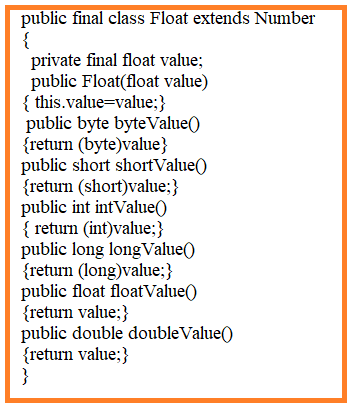
className: Integer



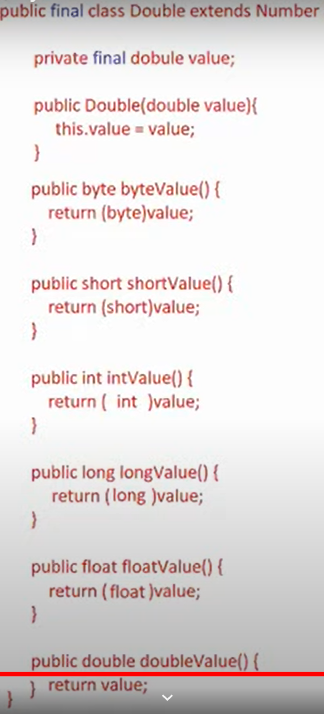
className:long



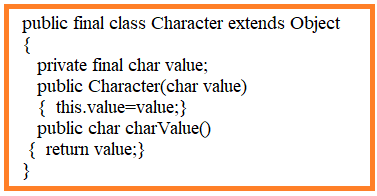
className:Float



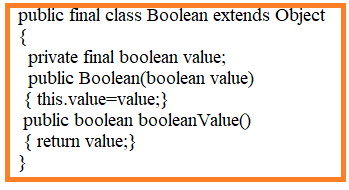
className: Double



className: Character



className:Boolean



Note:-Java5 onwards wrapperclass constructors are deprecated. Alternatively sunmicro system gave static method which is valueOf().

All wrapper class has toString() method. This method returns string representation of wrapper class object.

**CONVERTING PRIMITIVE DT INTO WRAPPER CLASS OBJECTS**

Example:- This program demonstrates that converting Primitive value into Wrapper class object.

public class sample

{

public static void main(String[] args)

{

//Byte a=new Byte(50); It lead to CE because compiler consider 50 as integer and look for Integer constructor in Byte class.

Byte a =new Byte((byte)50);

//Short s=new Short(5);It lead to CE because compiler consider 50 as integer and look for Integer constructor in Short class.

Short s= new Short((short)50);

Integer i = new Integer(50);

Long l= new Long(500);

Float f =new Float(25.5); //t does not leads to CE. Because Float class has double parameterized constructor also.

Float fg=new Float(2.55f);

Double d=new Double(50.5);

Character c =new Character('x');

//Character cv = new Character(97); It leads to CE. Because compiler look for Integer parameterized constructor in character class. It does not has.

Boolean b=new Boolean (true);

System.out.println(a);

System.out.println(s);

System.out.println(i);

System.out.println(l);

System.out.println(f);

System.out.println(fg);

System.out.println(d);

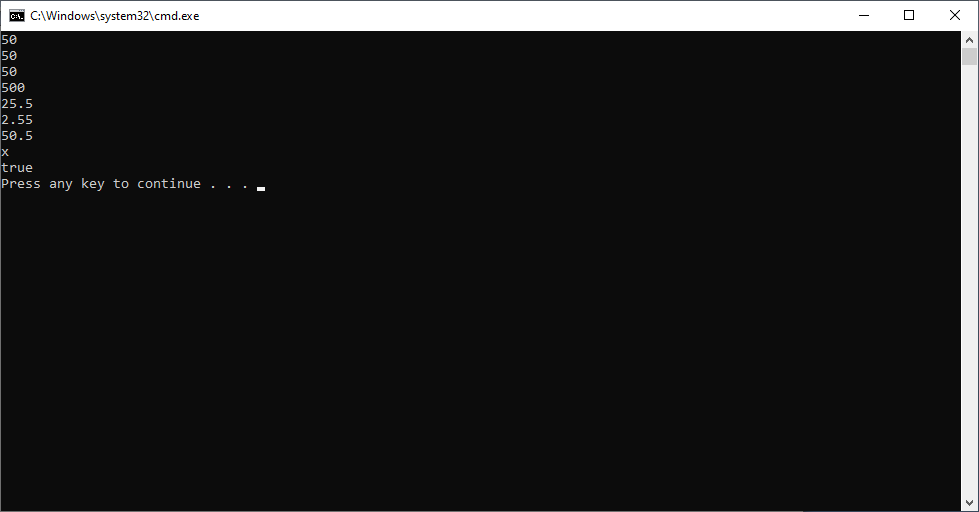
System.out.println(c);

System.out.println(b);

}

}

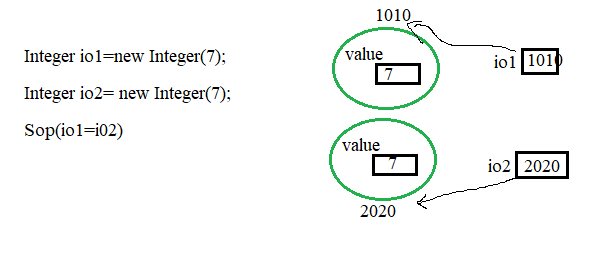
Output:-



Drawback in representing Primitive value as object using wrapper class constructor.

1. Every time object is created even though data is same.

Example:



Output: false.

Java5 onwards, In all wrapper classes all constructors are deprecreated. As alternative to constructors we got a static factory method valueOf(PDT). Its prototype is



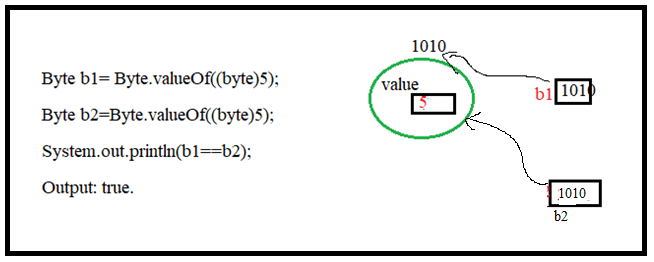
**Object Pooling**:- If object is already existed with given value, valueOf() method does not create new object. It returns existing object.

Note:- object pooling is only applied by default upto byte ranges values. -128 to 127.

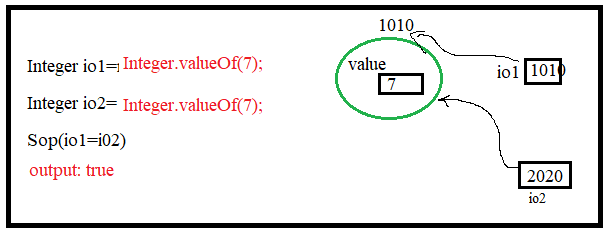
Note:-

1. Pooling is applied to only byte, Short, Integer,Long, Character and Boolean wrapper classes.
2. Pooling is not applied to Float & Double wrapper classes.

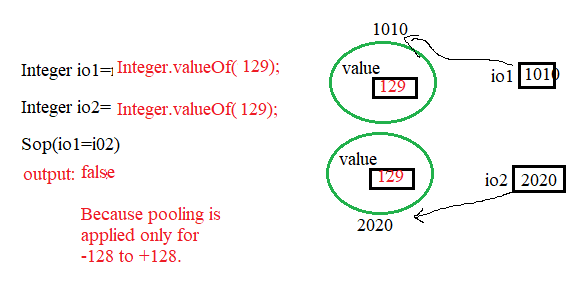
Example1:



Example2:



Example3:



Example: This program demonstrates that converting Primitive value into wrapper class object using “valueOf() “ method.

public class sample

{

public static void main(String[] args)

{

//Byte a=new Byte(50); It lead to CE because compiler consider 50 as integer and look for Integer constructor in Byte class.

Byte a =Byte.valueOf((byte)50);

//Short s=new Short(5);It lead to CE because compiler consider 50 as integer and look for Integer constructor in Short class.

Short s= Short.valueOf((short)50);

Integer i = Integer.valueOf(50);

Long l= Long.valueOf(500);

//Float f =Float.valueOf(25.5); it does leads to CE. Because Float class does not has double parameterized method also.

Float fg=Float.valueOf(2.55f);

Double d=Double.valueOf(50.5);

Character c =Character.valueOf('x');

//Character cv = new Character(97); It leads to CE. Because compiler look for Integer parameterized constructor in character class. It does not has.

Boolean b=Boolean.valueOf (true);

System.out.println(a);

System.out.println(s);

System.out.println(i);

System.out.println(l);

System.out.println(fg);

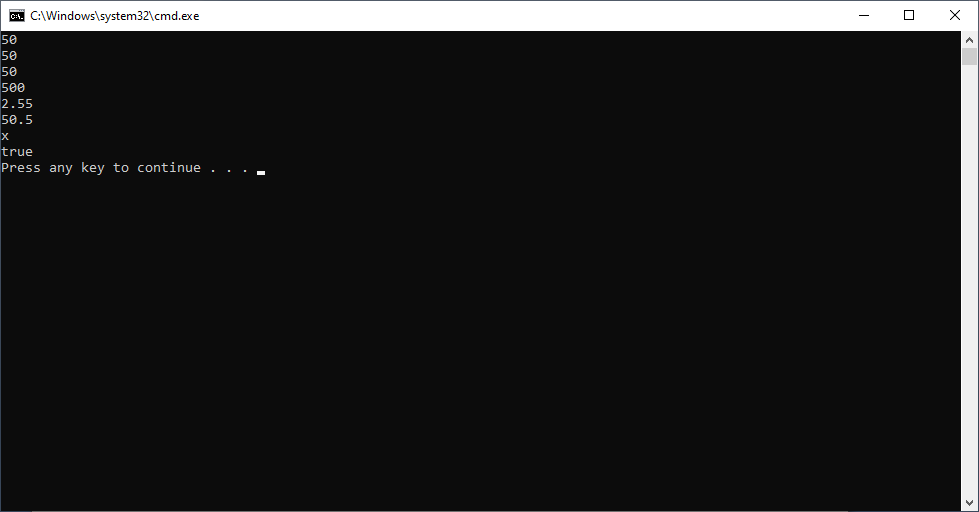
System.out.println(d);

System.out.println(c);

System.out.println(b);

}

}



**CONVERTING WRAPPER CLASS OBJECTS TO PRIMITIVE DT**

public class sample

{

public static void main(String[] args)

{

Byte b=Byte.valueOf((byte)10);

System.out.println("---Converting Byte into another primite data type using XXXvalue()----");

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

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

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

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

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

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

Short s=Short.valueOf((short)20);

System.out.println("---Converting Short into another primite data type using XXXvalue()----");

System.out.println(s.byteValue());

System.out.println(s.shortValue());

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

System.out.println(s.longValue());

System.out.println(s.floatValue());

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

Integer i=Integer.valueOf(25);

System.out.println("---Converting Integer into another primite data type using XXXvalue()----");

System.out.println(i.byteValue());

System.out.println(i.shortValue());

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

System.out.println(i.longValue());

System.out.println(i.floatValue());

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

Float f=Float.valueOf(23.4f);

System.out.println("---Converting Float into another primite data type using XXXvalue()----");

System.out.println(f.byteValue());

System.out.println(f.shortValue());

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

System.out.println(f.longValue());

System.out.println(f.floatValue());

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

System.out.println("---Converting Character wrapper object into Primitive value");

Character c=Character.valueOf('s');

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

System.out.println("---converting Boolean wrapper object into primitive value");

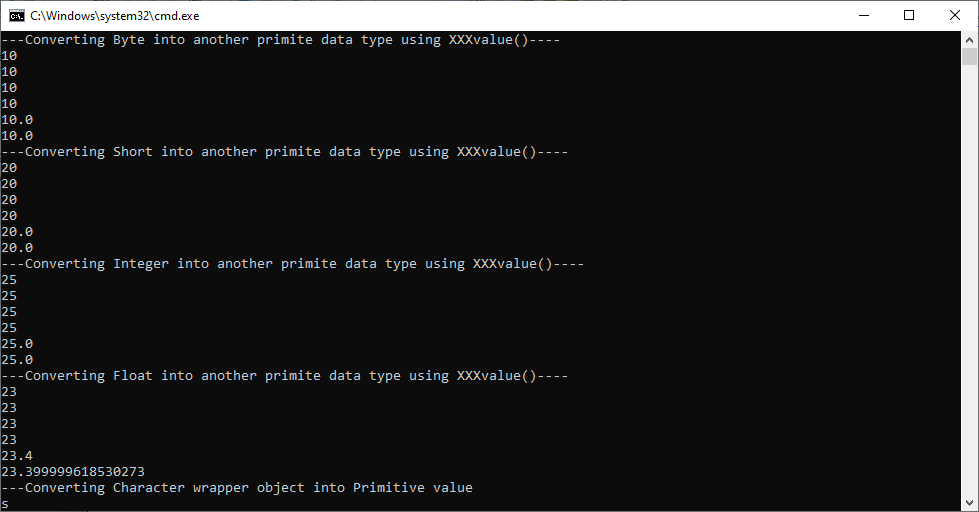
Boolean bx=Boolean.valueOf(true);

System.out.println(bx.booleanValue());

}

}

Output:



**AUTO BOXING & AUTO UN BOXING**

1. **AutoBoxing:-** converting primitive type to wrapper class object automatically is called Auto Boxing.

Example:

Integer io=50;

In above line , compiler converts int literal 50 to integer object as show below

Integer io=integer.valueOf(50);

So in “.class” file we do not have 50 as int litral we have it as Integer object. So JVM process the value 50 as integer object.

Example:-

public class sample

{

public static void main(String[] args)

{

Byte b= (byte)20;

Short s= (short)50;

Integer i=40;

Long l= 45L;

Float f= 34.4f;

Double d= 34.2;

Character c='x';

Boolean b1=true;

System.out.println(b);

System.out.println(s);

System.out.println(i);

System.out.println(l);

System.out.println(f);

System.out.println(d);

System.out.println(c);

System.out.println(b1);

}

}

Output:



2.AutoBoxing:- converting wrapper class object to primitive type automatically is caled Auto UnBoxing.

Example:

Int i=new Integer(50);

In the above line, compiler converts Integer object 50 to int value 50 as shown below

Int i=new Integer(50).intValue();

So, in .class file we do not have new Integer(50) as Integer object, since intValue() method is called. Its value 50 is returned and stored in “I” variable as int value. So JVM process the value 50 as int value.

public class sample

{

public static void main(String[] args)

{

byte b=Byte.valueOf((byte)10);

short s=Short.valueOf((short)20);

int i=Integer.valueOf(25);

long l=Long.valueOf(45);

float f=Float.valueOf(3.4f);

double d=Double.valueOf(5.4);

char c=Character.valueOf('c');

boolean b1=Boolean.valueOf(false);

System.out.println(b);

System.out.println(s);

System.out.println(i);

System.out.println(l);

System.out.println(f);

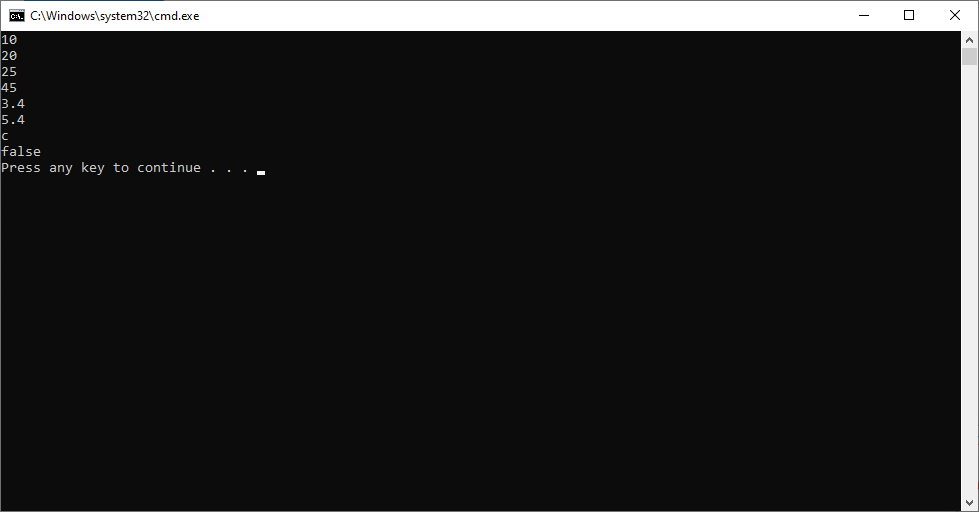
System.out.println(d);

System.out.println(c);

System.out.println(b1);

}

}



**WRAPPER CLASS CASTING**

Primitive data types are compatible, but their wrapper classes and arrays are not comparable, because wrapper classes are created as siblings. So we can not assign one type of wrapper class object to anothr type of wrapper class referenced variable either directly or via cast operator and als we can not compare them by using “==” or “!=” operators.

Example:

public class sample

{

public static void main(String[] args)

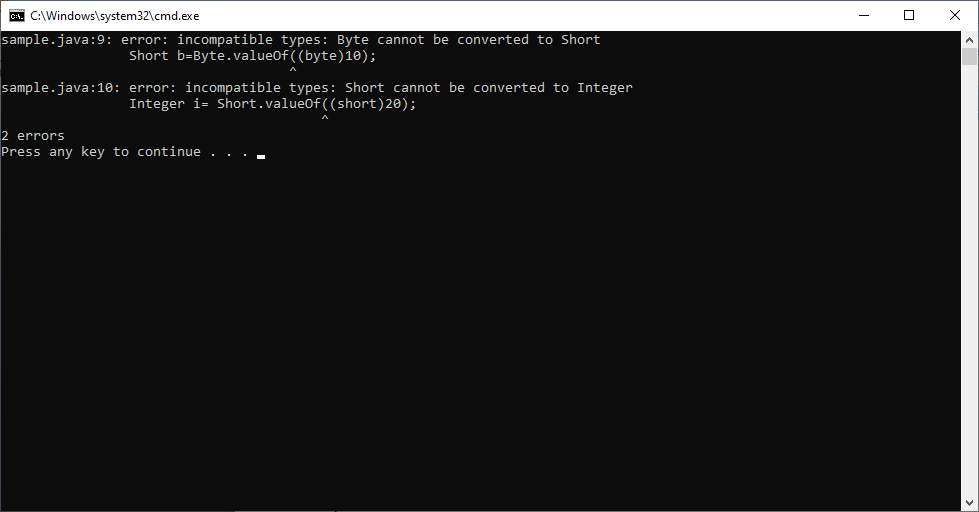
{

Short b=Byte.valueOf((byte)10);

Integer i= Short.valueOf((short)20);

}

}



**WRAPPER CLASS OBJECTS COMPARASION**

We can compare objects in two ways

1. Using “==” operator
2. Using equals() method.

“==” operator always compares two objects of same class using their references, where as “dequals()” compares two objects a wrapper class using their data.

Rule: Using “==” operator we can not compare incompatible classes objects, means those classes does not have inheritance relation, here compiler throws error. Whereas using equals() method we can compare incompatible objects, it will return false.

Example:-

public class sample

{

public static void main(String[] args)

{

Byte b=Byte.valueOf((byte)10);

Integer i= Integer.valueOf((short)20);

//System.out.println(b==i); //It leads to CE.

System.out.println(b.equals(i));

//System.out.println(b.equals((byte)10); It leads to CE Because equals expects wrapper class object as argument. But we passed PDT.

Byte b1=Byte.valueOf((byte)10);

System.out.println(b.equals(b1));

}

}

