Wrapper Classes

Wrapper Classes

- Java uses primitive types, such as int and char for performance reasons.
- The primitive data are passed by value and can not be directly passed by reference.
- There is no way for two methods to refer to the same instance of an int.
- There are some places where only objects are required to work with like collection classes.
- In that case we need to wrap the primitive type in a class.
- Java provides classes that correspond to each of the primitive types. These classes wrap, or encapsulate the primitive types within a class. So, they are known as wrapper classes.

Wrapper Classes

Superclass: Number

- This is an abstract class and is implemented by all the wrapper classes defined for numeric types byte, short, int, long, float and double.
- Following are the abstract methods inside Number

  byte byteValue() : Returns the value as byte
  float floatValue() : Returns the value as float
  double doubleValue() : Returns the value as double
  int intValue() : Returns the value as int
  short shortValue() : Returns the value as short
  long longValue() : Returns the value as long

- The calling object can be an object of Byte, Short, Integer, Long, Float or Double class
- The values returned by these methods can be rounded.

Wrapper Classes

Double and Float

Constructors (Float)

Float (double num)

Float (float num)

Float (String str) throws NumberFormatException

Constructors (Double)

Double (double num)

Double (String str) throws NumberFormatException

Wrapper Classes

Float: Some Important Methods

int compareTo(Float obj)

Compares the numeric value of two Float class objects and return 0, -ve value, or +ve value

static float parseFloat(String str)

Return float equivalent of the string str

String toString()

Converts Float object into String Object

static Float valueOf(String str)

Converts a string str that contains some float number into Float object

*** Similar methods for other Wrapper classes for different numeric types**

Wrapper Classes

Float: Some Important Methods

Using Constructor for creating Float object

float f=12.122f;

Float obj=new Float(f);

String str="12.122f";

Float obj=new Float(str);

Wrapper Classes

Boolean class

The Boolean class wraps a value of the primitive type boolean in an object. The Boolean class object contains a boolean type field that stores a primitive boolean

Constructor

Boolean(boolean value);

Boolean (String str)

Wrapper Classes

Boolean: Some Important methods

int compareTo(Boolean obj)

Compares the value of two Boolean class objects and return 0, -ve value, or +ve value

static boolean parseBoolean(String str)

Return boolean equivalent of the string str

String toString()

Converts Boolean object into String Object

static Boolean valueOf(String str)

Converts a string str that contains some boolean value into Boolean object

Wrapper Classes

Character Class

The Character class wraps a value of the primitive type char in an object.

Character class has only one constructor which accepts primitive data type

Character obj=new Character('A');

Wrapper Classes

Important methods of Character class

char charValue ( ):used to convert character object into character primitive

Character obj=new Character ('A');

char ch=obj.charValue();

int compareTo (Character obj):useful to compare two character objects

int x =obj1.compare (obj2);

if obj1==obj2,returns 0

if obj1<obj2,returns negative value

if obj1>obj2,returns positive value

String toString( ):converts character object into string object

Wrapper Classes

Important methods of Character class

static character value of (char ch)

convert a single character ch into character object

static boolean isDigit (char ch)

returns true if ch is a digit otherwise return false

static boolean isLetter(char ch)

returns true if ch is a letter

static boolean is UpperCase(char ch)

returns true if ch is a uppercase letter

static boolean is LowerCase(char ch)

returns true if ch is lower case letter

Wrapper Classes

Important methods of Character class

static boolean is SpaceChar(char ch)

returns true if ch is represents a white space

static boolean is LetterorDigit(char ch)

returns true if ch either a letter or digit

static char toUpperCase(char ch)

converts ch into uppercase

static char toLowerCase(char ch)

converts ch into lowercase

Wrapper Classes

Important methods of Character class

static boolean isSpaceChar(char ch)

returns true if ch is represents a white space

static boolean isLetterorDigit(char ch)

returns true if ch either a letter or digit

static char toUpperCase(char ch)

converts ch into uppercase

static char toLowerCase(char ch)

converts ch into lowercase

Wrapper Classes

- To convert a whole number into a decimal string, we can use versions of toString defined in various wrapper classes like Byte, Short, Integer or Long.

- The Integer and Long classes also provide the static methods toBinaryString(), toHexString(), and toOctalString(), which convert a vlaue into a binary, hexadecimal, or octal string, resepectively.

- Ex. Integer.toBinaryString(intWrapperObj);

  Integer.toBinaryString(integerPrimitiveData);

Wrapper Classes

Conversion Between Type Wrapper and Primitive Type (Boxing and Unboxing)

The process of encapsulating a value within an object is called **boxing**.

Integer iob = new Integer(10);

The process of extracting a value from a type wrapper is called **unboxing**.

int i = iob.intValue( );

Wrapper Classes

Autoboxing & Auto-unboxing

Autoboxing is the process by which a primitive type is automatically encapsulated (boxed) into its equivalent type wrapper whenever an object of that type is needed. There is no need to explicitly construct an object.

Auto-unboxing is the process by which the value of a boxed object is automatically extracted (unboxed) from a type wrapper when its value is needed. There is no need to call a method such as intValue( ) or doubleValue( ).

Wrapper Classes

Autoboxing & Auto-unboxing

```
Integer iob = 100;  //autobox an int

int i = iob;       //auto-unbox

System.out.println( i + ”  “ + iob); // displays 100 100
```

Wrapper Classes

Autoboxing & Auto-unboxing

Autoboxing/unboxing might occur when an argument is passed to a method, or when a value is returned by a method.

```
class BoxingDemo {

int fun(Integer v) {

        return v;

}

public static void main(String args[ ]) {

        Integer iob = fun(100);

        System.out.println(iob);

}

}
```

Wrapper Classes

Autoboxing & Auto-unboxing

Autoboxing/unboxing applies to expressions also.

Ex. Integer iob = 100;

++iob

Autoboxing and unboxing allows to mix different type of numeric objects in an expression.

Integer iob = 100;

Double dob = 98.6;

dob = iob + dob

System.out.println(dob) ; // 198.6

Wrapper Classes

Autoboxing & Auto-unboxing

Because of auto-unboxing, we can use Integer numeric objects to control a swtich statement.

Ex. Integer iob = 2;

Switch (iob) { …. } // iob is unboxed and its int value is obtained

Wrapper Classes

Autoboxing & Auto-unboxing

It applies to Boolean and Character wrapper types also.

Character ch = ‘x’ ; // box a char

char ch2 = ch; // unbox a char

Boolean b = true; // box a boolean

boolean b2 = b;

Because of unboxing a Boolean object can be used to check conditions wherever boolean values are used like in if statement and while conditions.

Ex. Boolean b = true; if (b) { …. }

Wrapper Classes

Autoboxing & Auto-unboxing

Autobox and auto-unbox **adds overhead** which is not present when the primitive type is used.

Bad use of autoboxing/unboxing:

```
Double a, b, c;

a = 5.0; b = 10.0;

c = Math.sqrt(a * a + b*b) ;        // This could be done using primitive types only.
```

In general, **we should restrict use of the type wrappers** to only those cases in which an object representation of a primitive type is required. Autoboxing / unboxing was not added to java as a “back door” way of eliminating primitive types.

Wrapper Classes

```
Integer iob1 = new Integer(31);
 String binaryData = Integer.toBinaryString(iob1);
 String hexData = Integer.toHexString(31);
  String octalData = Integer.toOctalString(31);
```