Wrapper Classes

In java wrapper classes commonly refers to the set of java classes that 'objectify' the primitive data types. For each primitive type, there is corresponding java wrapper class that represents that type. Example: the wrapper class for the int type is Integer class.

Wrapper Classes:

The java.lang package consists wrapper classes that correspond to each primitive type.

Primitive Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean
void	Void

Wrapper classes contains several methods that help manage the associated type.

Example:

```
String str = "1";
int num;
num = Integer.parseInt(str); // parseInt is static method that converts string into integer value.
```

Example:

```
class WrapperDemo {
        public static void main(String[] args) {
                Integer a = Integer.valueOf(100);
                String st = a.toString();
                System.out.println("Equivalent string is: " +st);
                System.out.println("Upper limit: " +Integer.MAX_VALUE);
                System.out.println("Lower limit: " +Integer.MIN_VALUE);
        }
}
```

Output:

```
Equivalent string is: 100
Upper limit: 2147483647
Lower limit: 2147483648
```

Autoboxing and Unboxing

The automatic conversion of primitive data types into it's equivalent wrapper type is known as boxing and the opposite operation (i.e. conversion from wrapper to primate) is known as unboxing. So, java programming doesn't need to write the conversion code.

Advantage:
→ No need of conversion between primitive type and wrapper type.

Example:

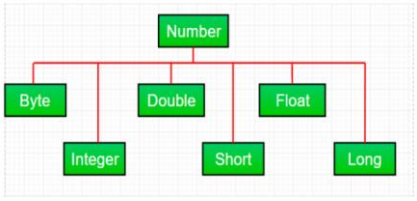
```
class BoxingExample {
public static void main(String[] args) {
int a = 50;
Integer a1 = new Integer(a); // boxing
Integer a3 = a; // boxing
}
}

class UnboxingExample {
public static void main(String[] args) {
Integer i = new Integer(50);
int a = i; // unboxing
System.out.println(a);
}
}
```

Need of Wrapper Class

- They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
- The classes in java.util package handles only objects and hence wrapper classes help in this case also.
- Data structures in the collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
- An object is needed to support synchronization in multithreading.

Number Class

There are mainly six sub-classes under Number class. These sub-classes define some useful methods which are used frequently while dealing with numbers.



Associated Methods

```
byte byteValue()
short shortValue()
int intValue()
long longValue()
float floatValue()
double doubleValue()
```

```
public class Test {
public static void main(String[] args) {
// Creating a Double Class object with value "6.9685"
Double d = new Double("6.9685");
// Converting this Double (Number) object to
// different primitive data types
byte b = d.byteValue();
short s = d.shortValue();
int i = d.intValue();
long 1 = d.longValue();
float f = d.floatValue();
double d1 = d.doubleValue();
System.out.println("value of d after converting it to byte : " + b);
System.out.println("value of d after converting it to short : " + s);
System.out.println("value of d after converting it to int : ” +i);
System.out.println("value of d after converting it to long : " + l);
System.out.println("value of d after converting it to float : " + f);
System.out.println("value of d after converting it to double : ” +d1);
}
}
```

```
value of d after converting it to byte : 6
value of d after converting it to short : 6
value of d after converting it to int : 6
value of d after converting it to long : 6
value of d after converting it to float : 6.9685
value of d after converting it to double : 6.9685
```

Methods Associated with Double

Method Name	Description	Example
toString()	Returns the string corresponding to the double value.	double b = 55.05; Double x = new Double(b); String s = Double.toString(x);
valueOf()	Returns the Double object initialized with the value provided.	String s = “45.55”; Double d = Double.valueOf(s);
parseDouble()	Returns double value by parsing the string. Differs from valueOf() as it returns a primitive double value and valueOf() return Double object.	String s = “45.55”; Double d = Double.parseDouble(s);

equals()	Used to compare the equality of two Double objects. This methods returns true if both the objects contains same double value.	If(double1.equals(double2))
		//true
		else // false

compareTo()	Used to compare two Double objects for numerical equality. Returns a value less than 0,0,value greater than 0 for less than,equal to and greater than.	If(double1.compareTo(double2))
		//true
		else // false

compare()	Used to compare two primitive double values for numerical equality.	If(Double.compare(double1,double2)==0) //equal
byteValue()	Returns a byte value corresponding to this Double Object.	byte b=double1.byteValue();
shortValue()	returns a short value corresponding to this Double Object.	short s=double1.shortValue();
intValue()	returns a int value corresponding to this Double Object.	int i=double1.intValue();
longValue()	returns a long value corresponding to this Double Object.	long l=double1.longValue();

doubleValue()	returns a double value corresponding to this Double Object.	double d=double1.doubleValue();
floatValue()	returns a float value corresponding to this Double Object.	float f=double1.floatValue();

Methods Associated with Float

Method Name	Description	Example
toString()	Returns the string corresponding to the float value.	float b = 55.05; Float x = new Float(b); String s = Float.toString(x);
valueOf()	Returns the Float object initialized with the value provided.	String s = "45.55"; Float d = Float.valueOf(s);
parseFloat()	Returns float value by parsing the string. Differs from valueOf() as it returns a primitive float value and valueOf() return Float object.	String s = "45.55"; Float d = Float.parseDouble(s);

equals()	Used to compare the equality of two Float objects. This methods returns true if both the objects contains same float value.	If(float1.equals(double2))
		//true
		else // false

compareTo()	Used to compare two Float objects for numerical equality. Returns a value less than 0,0,value greater than 0 for less than,equal to and greater than.	If(float1.compareTo(float2))
		//true
		else // false

compare()	Used to compare two primitive float values for numerical equality.	If(Float.compare(float1,float2)==0) //equal
byteValue()	Returns a byte value corresponding to this Float Object.	byte b=float1.byteValue();
shortValue()	returns a short value corresponding to this Float Object.	short s=float1.shortValue();
intValue()	returns a int value corresponding to this Float Object.	int i=float1.intValue();
longValue()	returns a long value corresponding to this Float Object.	long l=float1.longValue();
doubleValue()	returns a double value corresponding to this Float Object.	float d=float1.doubleValue();
floatValue()	returns a float value corresponding to this Float Object.	float f=float1.floatValue();

Methods Associated with Integer

1. **toString()**
2. **toHexString() :** Returns the string corresponding to the int value in hexadecimal form, that is it returns a string representing the int value in hex characters-[0-9][a-f].
3. **toOctalString() :** Returns the string corresponding to the int value in octal form, that is it returns a string representing the int value in octal characters-[0-7].
4. **toBinaryString() :** Returns the string corresponding to the int value in binary digits, that is it returns a string representing the int value in hex characters-[0/1].
5. **valueOf()**
6. **parseInt()**
7. **getInteger() :** returns the Integer object representing the value associated with the given system property or null if it does not exist.
8. **byteValue()**
9. **intValue()**
10. **doubleValue()**
11. **shortValue()**
12. **longValue()**
13. **floatValue()**

14. **equals()**
15. **compareTo()**
16. **compare()**

Methods Associated with Character

1. **boolean isLetter(char ch) :** This method is used to determine whether the specified char value(ch) is a letter or not. The method will return true if it is letter([A-Z],[a-z]), otherwise return false.

```
// Java program to demonstrate isLetter() method
public class Test {
  public static void main(String[] args) {
     System.out.println('A'));
     System.out.println('0'));
     }
}
// Output
true
false
```

2. **boolean isDigit(char ch) :** This method is used to determine whether the specified char value(ch) is a digit or not.

```
// Java program to demonstrate isDigit() method
public class Test {
  public static void main(String[] args) {
    // print false as A is character
    System.out.println('A'));
    System.out.println('0'));
    }
}
// Output
false
true
```

3. **boolean isWhitespace(char ch) :** It determines whether the specified char value(ch) is white space. A whitespace includes space, tab, or new line.

```
// Java program to demonstrate isWhitespace() method
public class Test {
  public static void main(String[] args) {
     System.out.println('A'));
     System.out.println(' '));
     System.out.println('\n'));
     System.out.println('\t'));
     // ASCII value of tab
     System.out.println(Character.isWhitespace(9));
     System.out.println('9'));
     }
}
// Output
false
true
true
true
true
false
```

4. **boolean isUpperCase(char ch) :** It determines whether the specified char value(ch) is uppercase or not.
5. **boolean isLowerCase(char ch) :** It determines whether the specified char value(ch) is lowercase or not.
6. **char toUpperCase(char ch) :** It returns the uppercase of the specified char value(ch).
7. **char toLowerCase(char ch) :** It returns the lowercase of the specified char value(ch).

```
// Java program to demonstrate toLowerCase() method
public class Test {
  public static void main(String[] args) {
     System.out.println(Character.toLowerCase('A'));
     System.out.println(Character.toLowerCase(65));
     System.out.println(Character.toLowerCase(48));
     }
}
```

8. **toString(char ch) :** It returns a String class object representing the specified character value(ch) i.e a one-character string.

```
// Java program to demonstrate toString() method
public class Test {
   public static void main(String[] args) {
   System.out.println(Character.toString('x'));
   System.out.println(Character.toString('Y'));
   }
}
```

Methods Associated with Boolean

1. **parseBoolean(String s) :** This method parses the string argument as a boolean. The boolean returned represents the value true if the string argument is not null and is equal, ignoring case, to the string "true", otherwise return false.

```
boolean b1 = Boolean.parseBoolean("True");
System.out.println(b1);
// Output : true
```

2. **booleanValue() :** This method returns the value of this Boolean object as a boolean primitive.

```
boolean b1 = Boolean.booleanValue("True");
System.out.println(b1);
// Output : true
```

3. **valueOf(boolean b) :** This method returns a Boolean instance representing the specified boolean value. If the specified boolean value is true, it returns Boolean.TRUE or if it is false, then this method returns Boolean.FALSE.

```
boolean b1 = true;
Boolean b3 = Boolean.valueOf(b1);
System.out.println(b3);
// Output : true
```

4. **valueOf(String s) :** This method returns a Boolean with a value represented by the specified string 's'. The Boolean returned represents a true value if the string argument is not null and is equal, ignoring case, to the string "true".
5. **toString(boolean b) :** This method returns a String object representing the specified boolean. If the specified boolean is true, then the string "true" will be returned, otherwise the string "false" will be returned.
6. **toString() :** This method returns a String object representing this Boolean's value. If this object represents the value true, a string equal to "true" is returned. Otherwise, the string "false" is returned.

7. **equals(Object obj) :** This method returns true iff the argument is not null and is a Boolean object that represents the same boolean value as this object.
8. **compareTo(Boolean b) :** This method "compares" this Boolean instance with passed argument 'b'.
9. **compare(boolean x, boolean y) :** This method is used to "compares" primitives boolean variables.

Methods Associated with other Wrapper Classes

// Same as Double, Float and Integer…

Java Primitive Data Type Autoboxing/Unboxing Boolean and Character Values

Java supplies wrappers for boolean and char. These are Boolean and Character. Autoboxing/unboxing applies to these wrappers, too. For example, consider the following program:

```
public class Main {
  public static void main(String[] args) {
  // Autobox/unbox a boolean.
  Boolean b = true;
  // Below, b is auto-unboxed when used in a conditional expression, such as an if. if (b)
  System.out.println("b is true");
  // Autobox/unbox a char.
  Character ch = 'x' ; // box a char
  char ch2 = ch; // unbox a char
  System.out.println("ch2 is"+ch2);
  }
}
// Output
b is true
ch2 is x
```

The auto-unboxing of b inside the if conditional expression. The conditional expression that controls an if statement must evaluate to type boolean. Because of auto-unboxing, the boolean value contained within b is automatically unboxed when the conditional expression is evaluated. With the autoboxing/unboxing, a Boolean object can be used to control an if statement. Because of auto-unboxing, a Boolean object can now also be used to control any of Java's loop statements. When a Boolean is used as the conditional expression of a while, for, or do/while, it is automatically unboxed into its boolean equivalent. For example, this is now perfectly valid code:

```
Boolean b;
// …
while(b) { // …
```

Autoboxing/Unboxing in Expressions

In general, autoboxing and unboxing take place whenever a conversion into an object or from an object is required. This applies to expressions. Within an expression, a numeric object is automatically unboxed. The outcome of the expression is reboxed, if necessary. For example, consider the following program:

```
// Autoboxing/unboxing occurs inside expressions.

class AutoBox3 {
  public static void main(String args[]) {
    Integer iOb, iOb2;
    int i;

    iOb = 99;
    System.out.println("Original value of iOb: " + iOb);

    // The following automatically unboxes iOb,
    // performs the increment, and then reboxes
    // the result back into iOb.
    ++iOb;
    System.out.println("After ++iOb: " + iOb);

    // Here, iOb is unboxed, its value is increased by 10,
    // and the result is boxed and stored back in iOb.
    iOb += 10;
    System.out.println("After iOb += 10: " + iOb);

    // Here, iOb is unboxed, the expression is
    // evaluated, and the result is reboxed and
    // assigned to iOb2.
    iOb2 = iOb + (iOb / 3);
    System.out.println("iOb2 after expression: " + iOb2);

    // The same expression is evaluated, but the
    // result is not reboxed.
    i = iOb + (iOb / 3);
    System.out.println("i after expression: " + i);
  }
}
```

Autoboxing/ unboxing occurs in expressions.

The output is shown here:

```
Original value of iOb: 99
After ++iOb: 100
After iOb += 10: 110
iOb2 after expression: 146
i after expression: 146
```

Autoboxing and methods

In addition to the simple case of assignments, autoboxing automatically occurs whenever a primitive type must be converted into an object, and auto-unboxing takes place whenever an object must be converted into a primitive type. Thus, autoboxing/unboxing might occur when an argument is passed to a method or when a value is returned by a method. For example, consider the following:

```
// Autoboxing/unboxing takes place with
// method parameters and return values.

class AutoBox2 {
  // This method has an Integer parameter.
  static void m(Integer v) {
    System.out.println("m() received " + v);
  }
  // This method returns an int.
  static int m2() {
    return 10;
  }
  // This method returns an Integer.
  static Integer m3() {
    return 99; // autoboxing 99 into an Integer.
  }

  public static void main(String args[]) {

    // Pass an int to m(). Because m() has an Integer
    // parameter, the int value passed is automatically boxed.
    m(199);



    // Here, iOb receives the int value returned by m2().
    // This value is automatically boxed so that it can be
    // assigned to iOb.
    Integer iOb = m2();
    System.out.println("Return value from m2() is " + iOb);

    // Next, m3() is called. It returns an Integer value
    // which is auto-unboxed into an int.
    int i = m3();
    System.out.println("Return value from m3() is " + i);

    // Next, Math.sqrt() is called with iOb as an argument.
    // In this case, iOb is auto-unboxed and its value promoted to
    // double, which is the type needed by sqrt().
    iOb = 100;
    System.out.println("Square root of iOb is " + Math.sqrt(iOb));
  }
}
```

Receives an **Integer**.

Returns an **int**.

Returns an **Integer**.