Programming in Java

Wrapper Classes

Introduction

- Most of the objects collection store objects and not primitive types.

- Primitive types can be used as object when required.

- As they are objects, they can be stored in any of the collection and pass this collection as parameters to the methods.

Wrapper Class

- Wrapper classes are classes that allow primitive types to be accessed as objects.

- Wrapper class is wrapper around a primitive data type because they "wrap" the primitive data type into an object of that class.

What is Wrapper Class?

- Each of Java's eight primitive data types has a class dedicated to it.

- They are one per primitive type: Boolean, Byte, Character, Double, Float, Integer, Long and Short.

- Wrapper classes make the primitive type data to act as objects.

Primitive Data Types and Wrapper Classes

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

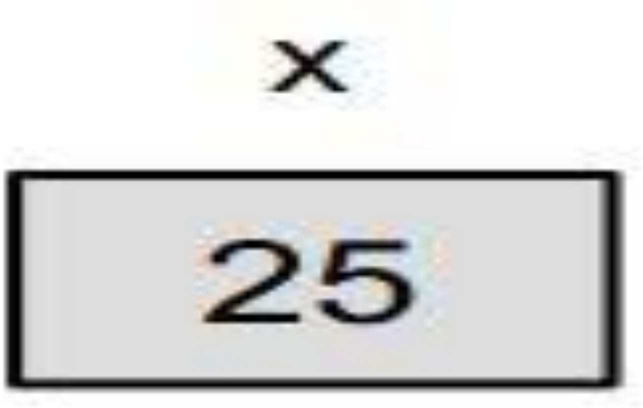
Difference b/w Primitive Data Type and Object of a Wrapper Class

- The following two statements illustrate the difference between a primitive data type and an object of a wrapper class:

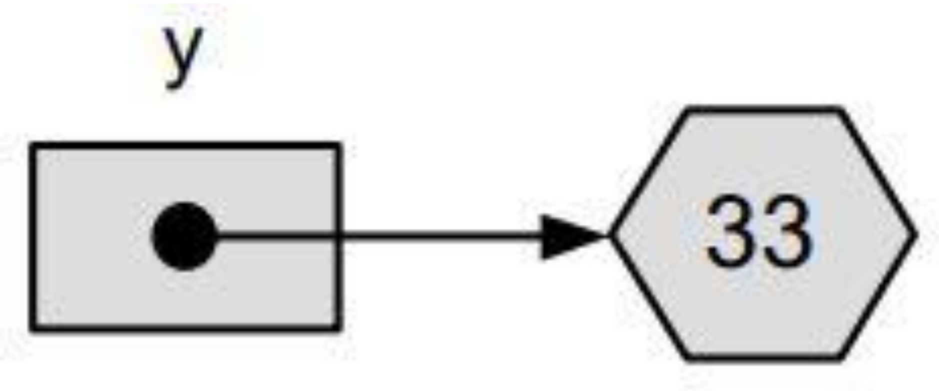
```
int x = 25;

Integer y = new Integer(33);
```

- The first statement declares an int variable named x and initializes it with the value 25.



- The second statement instantiates an Integer object. The object is initialized with the value 33 and a reference to the object is assigned to the object variable y.



- Clearly x and y differ by more than their values:

  x is a variable that holds a value;

  y is an object variable that holds a reference to an object.

- So, the following statement using x and y as declared above is not allowed:

```
int z = x + y;  // semantically wrong!
```

- The data field in an Integer object is only accessible using the methods of the Integer class.

- One such method is intValue() method which returns an int equal to the value of the object, effectively "unwrapping" the Integer object:

```
int z = x + y.intValue();  // OK!
```

What is the need of Wrapper Classes?

- Wrapper classes are used to be able to use the primitive data-types as objects.
- Many utility methods are provided by wrapper classes.

To get these advantages we need to use wrapper classes.

Conversion to Various Bases

- class Wrapper2
- {
- public static void main(String [] args)
- {
- //Integer i=new Integer("25");
- //Integer j=new Integer("12");
- //System.out.println(i+j);
- //Integer.parseInt(String s, int radix)
- int x=Integer.parseInt("1000", 2);
- System.out.println(x);
- String a=Integer.toString(15, 16);
- System.out.println(a);
- }
- }

Boxing and Unboxing

- The wrapping is done by the compiler.
- if we use a primitive where an object is expected, the compiler boxes the primitive in its wrapper class.
- Similarly, if we use a number object when a primitive is expected, the compiler un-boxes the object.

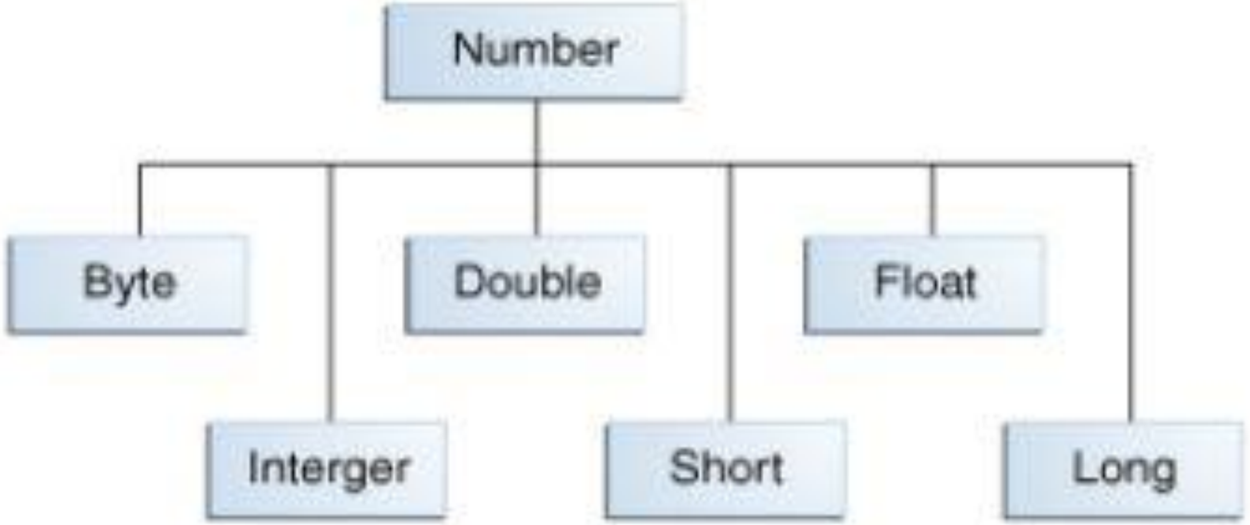
Example of boxing and unboxing:

- Integer x, y;   x = 12; y = 15;   System.out.println(x+y);
- When x and y are assigned integer values, the compiler boxes the integers because x and y are integer objects.
- In the println() statement, x and y are unboxed so that they can be added as integers.

Numeric Wrapper Classes

- All of the numeric wrapper classes are subclasses of the abstract class Number .

- Short, Integer, Double and Long implement Comparable interface.



Features of Numeric Wrapper Classes

- All the numeric wrapper classes provide a method to convert a numeric *string into a primitive value*.

public static type parseType (String Number)

public static type parseType (String Number, int Radix)

- parseInt()
- parseFloat()
- parseDouble()
- parseLong()
- parseShort()
- parseByte()

Features of Numeric Wrapper Classes

- All the wrapper classes provide a static method toString to provide the *string representation of the primitive values.*

public static String toString (type value)

Example:

public static String toString (int a)

Features of Numeric Wrapper Classes

- All numeric wrapper classes have a static method valueOf, which is used to *create a new object initialized to the value* represented by the specified string.

public static DataType valueOf (String s)

Example:

```
Integer i = Integer.valueOf (“135”);
Double d = Double.valueOf (“13.5”);
```

Methods implemented by subclasses of Number

- Compares this Number object to the argument.

  int compareTo(Byte anotherByte)
  int compareTo(Double anotherDouble)
  int compareTo(Float anotherFloat)
  int compareTo(Integer anotherInteger)
  int compareTo(Long anotherLong)
  int compareTo(Short anotherShort)

- returns int after comparison (-1, 0, 1).

Methods implemented by subclasses of Number

boolean equals(Object obj)

- Determines whether this number object is equal to the argument.
- The methods return true if the argument is not null and is an object of the same type and with the same numeric value.

Character Class

- Character is a wrapper around a char.

- The constructor for Character is :

```
Character(char ch)
```

Here, ch specifies the character that will be wrapped by the Character object being created.

- To obtain the char value contained in a Character object, call charValue( ), shown here:

```
char charValue( );
```

- It returns the encapsulated character.

Boolean Class

- Boolean is a wrapper around boolean values.

- It defines these constructors:

  Boolean(boolean boolValue)

  Boolean(String boolString)

- In the first version, boolValue must be either true or false.

- In the second version, if boolString contains the string “true” (in uppercase or lowercase), then the new Boolean object will be true. Otherwise, it will be false.

- To obtain a boolean value from a Boolean object, use instance method booleanValue( ), shown here:

  boolean booleanValue( )

- It returns the boolean equivalent of the invoking object.
- To create object from Boolean value use static method of Boolean class

valueOf(boolean b).