Java Wrapper Classes

A Wrapper class is a class whose object **wraps** or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need of Wrapper Classes

To 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.

Use of Java Wrapper classes:

We must use wrapper classes when working with Collection objects, such as `ArrayList`, `Map`, `HashTable ect`, where primitive types cannot be used (the list can only store objects).

Example:

```
ArrayList<Integer> myNumbers = new ArrayList<Integer>();

Hashtable<Integer, String> hashtable = new Hashtable<Integer,
String>();

Map<Integer, String> map = new HashMap<Integer, String>();
```

Wrapper classes provide a way to use primitive data types (**int**, **boolean**, etc..) as objects. The table below shows the primitive type and the equivalent wrapper class:

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

Creating Wrapper Objects or wrapper class objects:

To create a wrapper class object -

1. Use the wrapper class instead of the primitive type.

```
Integer myInt = 5;

Double myDouble = 5.99;

Character myChar = 'A';
```

2. Use the class constructor and pass value into the constructor. For example, Integer() is the Integer class constructor and we are passing the value 5 into it so that **i** holds the value **5.**

```
Integer i = new Integer(5);
Double d = new Double(5.99);
Character c = new Character('A');
```

To get the value, you can just print the objects (in bold letters):

```
System.out.println(myInt);
System.out.println(myDouble);
System.out.println(myChar);
```

To get the values from the objects of the Wrapper class, we use corresponding `Value()` methods:

Primitive Data Type	Wrapper Class	Method
byte	Byte	`byteValue()`
short	Short	`shortValue()`
int	Integer	`intValue()`
long	Long	`longValue()`
float	Float	`floatValue()`
double	Double	`doubleValue()`
boolean	Boolean	`booleanValue()`
char	Character	`charValue()`

`toString()` method is used to convert wrapper objects to strings. Example:

```
// creating a Wrapper class object myInt
Integer myInt = 100;

// converting the object to String
String myString = myInt.toString();

// print value of myInt (value is 100 is a String now, not an Integer)
System.out.println(myInt);
```

Converting an Integer to a String:

```
package collectionsdemo;
public class IntegerToStringConversion {
	public static void main(String[] args) {
		int x = 5;
		String s = Integer.toString(x);
		System.out.println(s);
	}
}
```

Converting a String to an Integer:

```
package collectionsdemo;
public class Converts {
      public static void main(String[] args) {
      String number = "100"; // declaring a String with a value 100
      int n = Integer.valueOf(number); // converting a String to an int
      System.out.println(n);
 // to check if n is an integer, lets add 123 with n
 System.out.println(n + 123); // the output is a sum of n +123 = 223
      }
}
```

Another way of Converting a String to an Integer:

```
public class TestList {
      public static void main(String[] args) {
            String s = new String ("1234");
            int i = Integer.valueOf(s); // converts string to int
            System.out.println(i);
            System.out.println(i+123456);
      }

}
```

Further study: https://www.geeksforgeeks.org/wrapper-classes-java/