# Lecture - Wrapper classes

## Wrapper classes ↗ change character value to integer

Variables of primitive types are not Java objects.  That is, they are not of a class derived from class Object.

That means we cannot have, for example, a java List of integers, because the List class can only have lists of Object elements.

However, java provides classes that behave like the primitive types:

```
boolean -> Boolean
byte    -> Byte
short   -> Short
long    -> Long
float   -> Float
double  -> Double
```

Each has a capital letter, following the Java convention for naming classes.

There are two more wrappers, which have names that aren't just capitalizations:

```
char -> Character
int  -> Integer
```

(Why not follow the same naming pattern?  I have no idea.)

Wrapper classes also have many useful constants and static methods

e.g., Integer.decode(String s) converts a decimal string to an Integer.

## Boxing

Boxing is the process of going from a value of a primitive type to an object of its wrapper class

- To convert a primitive value to an "equivalent" class type value, create an object of the corresponding wrapper class using the primitive value as an argument
- The new object will contain an instance variable that stores a copy of the primitive value
- Unlike most other classes, a wrapper class does not have a no-argument constructor

```
Integer integerObject = new Integer(42);
```

## Unboxing

Unboxing: the process of going from an object of a wrapper class to the corresponding value of a primitive type.  The methods for this are

```
  Boolean.booleanValue()
     Byte.byteValue()
    Short.shortValue()
  Integer.intValue()
    Float.floatValue()
   Double.doubleValue()
Character.charValue()
```

None of these methods take an argument

```
int i = integerObject.intValue();
```

## Automatic boxing and unboxing

Manually inserting boxing and unboxing code makes coding slower and makes code harder to read.

Starting with version 5.0, Java can automatically do boxing and unboxing.

Instead of creating a wrapper class object using the new operation (as shown before), it can be done as an automatic type cast:

```
Integer integerObject = 42;
```

Instead of having to invoke the appropriate method (such as intValue) to convert from an object of a wrapper class to a value of its associated primitive type, the primitive value can be recovered automatically

```
int i = integerObject;
```

Try it yourself.  Remove the boxing and unboxing in the following code.  (The "(double)" in line 7 is called a cast.)

```
class Main {
    public static void main (String[] arg) {
        Boolean b = new Boolean(true);
        System.out.println("Boolean is " + b.booleanValue());

        Integer i = new Integer (4);
        Double d = new Double (Math.sqrt((double)(i.intValue())));
        System.out.println("Double is " + d.doubleValue());
    }
}
```

```
Boolean is true
 Double is 2.0
```

## Static methods

Wrapper classes have static methods that convert a correctly formed string representation of a number to the
number of a given type

- `Integer.parseInt ()`
- `Long.parseLong ()`
- `Float.parseFloat ()`
- `Double.parseDouble ()`

Wrapper classes also have static methods that convert from a numeric value to a string representation of the value

- `Double.toString(123.99);`   returns the string value "123.99"

The Character class contains a number of static methods that are useful for string processing.  See Display 5.8 in the textbook.

```
public static char toUpperCase(char argument)
public static char toLowerCase(char argument)
```

Replace lower case characters by upper case equivalents, or vice versa

```
public static boolean isUpperCase(char argument)
public static boolean isLowerCase(char argument)
```

Returns true if the argument is an upper-case letter, and false otherwise.

The following return true if argument is...

```
public static boolean isWhitespace(char argument) // Whitespace (space, tab \t, new line \n)
public static boolean isLetter(char argument)     // A letter a-z, A-Z, accented chars
public static boolean isDigit(char argument)
public static boolean isLetterOrDigit(char argument)
```

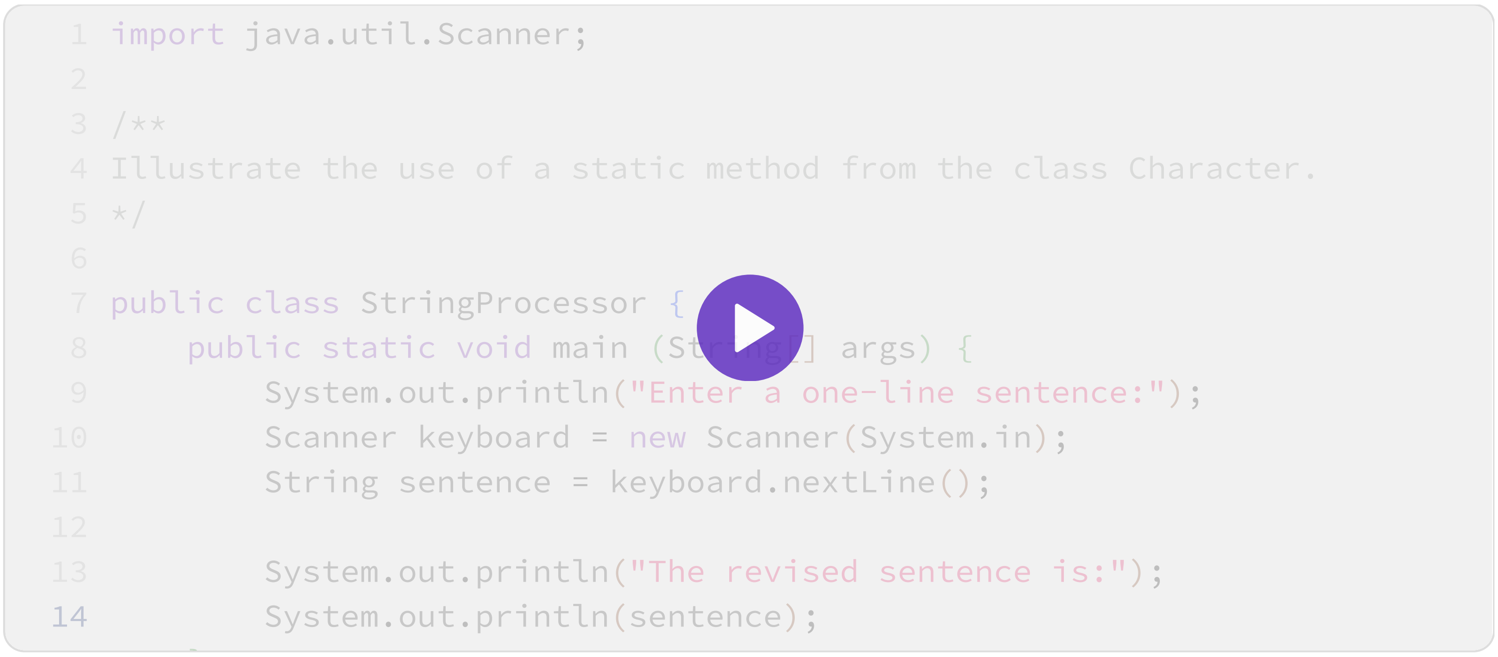
## Advanced: Unicode

`isUpperCase` and `isLowerCase` only return `true` for Roman letters (a-z,A-Z, and accented versions) and Greek letters.  They both return false for characters from alphabets that do not have upper and lower case.

`isLetter` returns `true` for unicode letters from other languages, such as Hindi and Chinese. Combining this with testing for both `isUpperCase` and `isLowerCase` allows you to distinguish between Roman/Greek letters and other scripts.

`isDigit` returns `true` for characters like ▯ that are only digits but not letters, but not for Chinese digits like '二' that are also classed as "letter", nor for decorated digits like ①.

## Example: Sentence case



```java
import java.util.Scanner;

/**
Illustrate the use of a static method from the class Character.
*/

public class StringProcessor {
    public static void main (String[] args) {
        System.out.println("Enter a one-line sentence:");
        Scanner keyboard = new Scanner(System.in);
        String sentence = keyboard.nextLine();

        System.out.println("The revised sentence is:");
        System.out.println(sentence);
```

## ...your own extension to StringProcessor

Here is the same string processor example. Now it is your turn.

1. Modify the code to add a "." at the end of the sentence if it ends with a letter or a digit. You can find the length of a string using the length() method.
2. Print "This sentence contains a number" (once only) if any of the characters is a digit.
3. **Advanced:** Style guides recommend spelling out the integers 0-9. Replace any 1-digit number by the text form of it. For example "1" -> "one", "0"->"zero". But leave two-digit numbers unchanged.