Character Class

Normally, when we work with characters, we use primitive data types char.

Examples

```
char ch = 'a';

// Unicode for uppercase Greek omega character
char uniChar = '\u039A';

// an array of chars
char[] charArray ={ 'a', 'b', 'c', 'd', 'e' };
```

Character Class

- However in development, we come across situations where we need to use objects instead of primitive data types. In order to achieve this, Java provides wrapper class Character for primitive data type char.

- The Character class offers a number of useful class (i.e., static) methods for manipulating characters. You can create a Character object with the Character constructor

  Character ch = new Character('a');

- The Java compiler will also create a Character object for you under some circumstances. For example, if you pass a primitive char into a method that expects an object, the compiler automatically converts the char to a Character for you. This feature is called autoboxing or unboxing, if the conversion goes the other way.

Character Class

Example

```
// Here following primitive char 'a'
// is boxed into the Character object ch
Character ch = 'a';

// Here primitive 'x' is boxed for method test,
// return is unboxed to char 'c'
char c = test('x');
```

Character Methods

Method & Description
isLetter() **Determines whether the specified char value is a letter.**
isDigit() **Determines whether the specified char value is a digit.**
isWhitespace() **Determines whether the specified char value is white space.**
isUpperCase() **Determines whether the specified char value is uppercase.**

Character Methods

Method & Description
isLowerCase() **Determines whether the specified char value is lowercase.**
toUpperCase() **Returns the uppercase form of the specified char value.**
toLowerCase() **Returns the lowercase form of the specified char value.**
toString() **Returns a String object representing the specified character value that is, a one-character string.**