Programming 1

Week 10 – Wrapper Classes

Introduction to Wrapper Classes

- Java provides 8 primitive data types.
- They are called “primitive” because they are not created from classes.
- Wrapper classes provide a way to use primitive data types (int, boolean, etc..) as objects.
- Java provides wrapper classes for all of the primitive data types.
- **Wrapper classes help in conversion from one datatype to another datatypes**
- The wrapper classes are part of `java.lang` so to use them, there is no `import` statement required.

Wrapper Classes

- Wrapper classes allow you to create objects to represent a primitive.
- Wrapper classes are immutable, which means that once you create an object, you cannot change the object’s value.
- Since you're now working with objects, you can use certain methods to get information about the specific object.
- To get the value stored in an object you must call a method.
- Wrapper classes provide static methods that are very useful

Data Type Wrappers

- Java provides wrapper classes for all of the primitive data types.
- The primitive wrapper classes are:

Wrapper Class	Numeric Primitive Type It Applies To
`Byte`	`byte`
`Double`	`double`
`Float`	`float`
`Integer`	`int`
`Long`	`long`
`Short`	`short`
`Character`	`Char`
`Boolean`	`Boolean`

Creating a Wrapper Object

- To create objects from these wrapper classes, you can pass a value to the constructor:

```
Integer number = new Integer(7);
```

- You can also assign a primitive value to a wrapper class object:

```
Integer number;
number = 7;
```

The `Parse` Methods (1 of 2)

- Any string containing a number, such as “127.89”, can be converted to a numeric data type.
- Each of the numeric wrapper classes has a static method that converts a string to a number.
  - The `Integer` class has a method that converts a `String` to an `int`,
  - The `Double` class has a method that converts a `String` to a `double`,
  - etc.
- These methods are known as **parse methods** because their names begin with the word “parse.”

The `Parse` Methods (2 of 2)

```
// Store 1 in bVar.
byte bVar = Byte.parseByte("1");
// Store 2599 in iVar.
int iVar = Integer.parseInt("2599");
// Store 10 in sVar.
short sVar = Short.parseShort("10");
// Store 15908 in lVar.
long lVar = Long.parseLong("15908");
// Store 12.3 in fVar.
float fVar = Float.parseFloat("12.3");
// Store 7945.6 in dVar.
double dVar = Double.parseDouble("7945.6");
```

- The parse methods all throw a `NumberFormatException` if the `String` object does not represent a numeric value.

The `toString` Methods

- Each of the numeric wrapper classes has a static `toString` method that converts a number to a string.
- The method accepts the number as its argument and returns a string representation of that number.

```
int i = 12;
double d = 14.95;
String str1 = Integer.toString(i);
String str2 = Double.toString(d);
```

MIN_VALUE and MAX_VALUE

- The numeric wrapper classes each have a set of static final variables
  - `MIN_VALUE` and
  - `MAX_VALUE`.
- These variables hold the minimum and maximum values for a particular data type.

```
System.out.println("The minimum value for an "
                   + "int is "
                   + Integer.MIN_VALUE);
System.out.println("The maximum value for an "
                   + "int is "
                   + Integer.MAX_VALUE);
```

Character Testing and Conversion With The `Character` Class

- The `Character` class allows a char data type to be **wrapped** in an object.
- The `Character` class provides methods that allow easy testing, processing, and conversion of character data.

The `Character` Class Static Methods

Method	Description
isUpperCase()	Tests if character is uppercase
toUpperCase()	Returns the uppercase equivalent of the argument; no change is made if the
	argument is not a lowercase letter
isLowerCase()	Tests if character is lowercase
toLowerCase()	Returns the lowercase equivalent of the argument; no change is made if the argument is not an uppercase letter
isDigit()	Returns true if the argument is a digit (0-9) and false otherwise
isLetter()	Returns true if the argument is a letter and false otherwise
isLetterOrDigit()	Returns true if the argument is a letter or digit and false otherwise
isWhitespace()	Returns true if the argument is whitespace and false otherwise; this
	includes the space, tab, newline, carriage return, and form feed

Commonly used methods of the Character class

Character Testing and Conversion With The **`Character`** Class

- The `Character` class provides two methods that will change the case of a character.

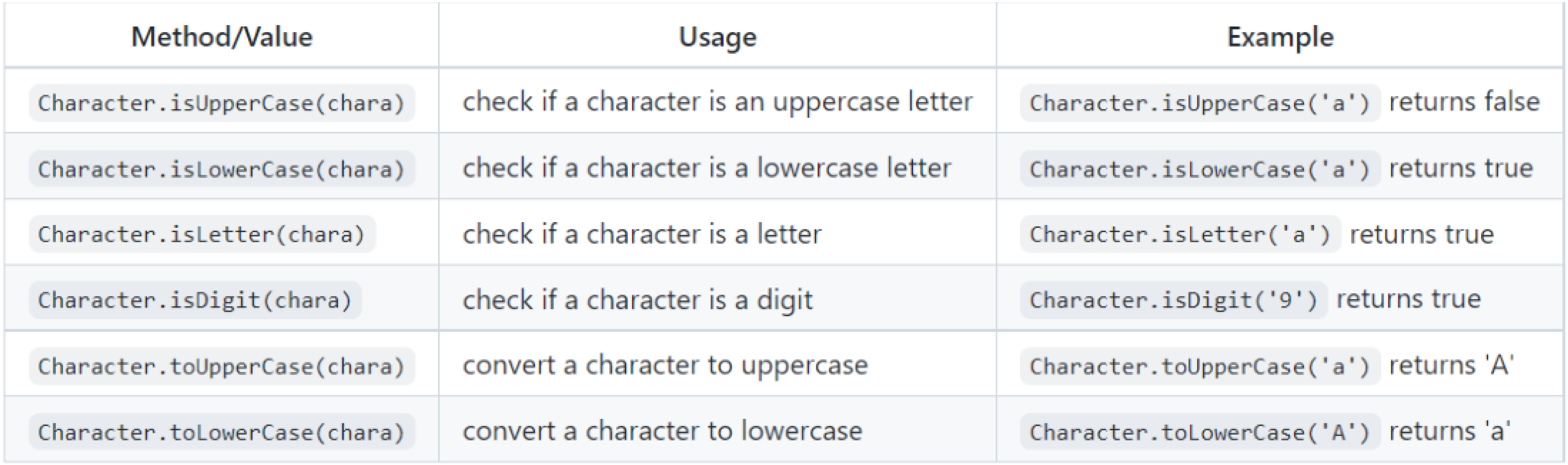
```
boolean Character.toLowerCase(char ch)
```

Returns the lowercase equivalent of the argument passed into *ch*.

```
boolean Character.toUpperCase(char ch)
```

Returns the uppercase equivalent of the argument passed into *ch*.

`Character` Methods Example



Method/Value	Usage	Example
`Character.isUpperCase(chara)`	check if a character is an uppercase letter	`Character.isUpperCase('a')` returns false
`Character.isLowerCase(chara)`	check if a character is a lowercase letter	`Character.isLowerCase('a')` returns true
`Character.isLetter(chara)`	check if a character is a letter	`Character.isLetter('a')` returns true
`Character.isDigit(chara)`	check if a character is a digit	`Character.isDigit('9')` returns true
`Character.toUpperCase(chara)`	convert a character to uppercase	`Character.toUpperCase('a')` returns 'A'
`Character.toLowerCase(chara)`	convert a character to lowercase	`Character.toLowerCase('A')` returns 'a'