Wrapper classes

An instance of class `Integer` contains a field of type **`int`**. We haven't given the name of the field because we don't know it. But there is a getter method for it, `intValue()`. In fact, one can obtain the **int** value as a primitive value of other types —**`byte`**, **`short`**, and so on— and also as a `String`, using these functions:

```
intValue()      byteValue()      shortValue()      longValue()
floatValue()    doubleValue()
toString()
```

But there is no setter method, so the field can't be changed. We say that it is *immutable*.

Instance function `equals` yields true iff its parameter is an object of class `Integer` and the parameter's wrapped value equals the instance's wrapped value.

```
Integer b= new Integer(5);
b.equals(new Double(5))         is      false
b.equals(new Integer(6))        is      false
b.equals(new Integer(5))        is      true
```

Using wrapper class Integer

`Integer` is called a *wrapper class* for type **`int`**, because an instance wraps, or contains a single integer of type **`int`**, like you wrap a sandwich in saran wrap or cellophane or napkin. We can use an assignment statement to wrap an integer in an instance of `Integer`:

```
Integer x= new Integer(25);
```

So, we have one reason for the existence of type `Integer`:

Reason for wrapper class `Integer`: to allow us to handle a primitive **`int`** value like an object.

In the next web lecture, we use this technique to put an integer into an instance of class `ArrayList`.

Constants of class Integer

A second reason for class `Integer` is:

Reason for wrapper class `Integer`: to provide useful constants and functions that deal with **`ints`**.

Thus, class `Integer` provides constants for the minimum and maximum values of type **`int`**. Since these are static fields, one can reference them using the name of the class —we will introduce static fields later.

```
Integer.MIN_VALUE
Integer.MAX_VALUE
```

Static functions of class Integer

Class Integer has a number of static functions that deal with **ints**. For example, one can find the binary, octal, and hexadecimal representations of integers as strings:

```
Integer.toBinaryString(25)     is      "11001"
Integer.toOctalString(25)      is      "31"
Integer.toHexString(25)        is      "19"
```

Finally, Integer has a static function to translate a string that contains an integer into an **`int`**:

```
Integer.parseInt("25")         is      25
```

The argument of a `parseInt` call must contain only digits, possibly with a preceding minus sign. Not even blanks are allowed.

```
Integer.parseInt("25 ")        produces an error
```

Summary

In summary, wrapper class `Integer` allows us to handle an **`int`** value as an object and provides some useful functions that deal with **int** values. If you want to get the minimum or maximum **int** value, or you want to see the binary, octal, or hexadecimal representation of an integer, or you have a string that has to be translated to an integer, then turn to class Integer for help.