Section Eight – Wrapper Classes in Java

In this section the Developer will learn…

- About the Wrapper classes available to store numbers

Wrapper Classes - Introduction

All primitive datatypes have Classes available which contain methods relating to their particular type, these methods will allow the Developer to perform more complex tests, change values between datatypes and create the datatypes (constructors) with more complex values (that is from different datatypes).

These classes are called 'Wrappers' and effectively support each primitive datatype with a series of extended facilities. The datatypes used so far are referred to as 'primitive datatypes'.

The following slide gives the primitive and its equivalent wrapper class …

Wrapper Classes - Introduction

Primitive	Wrapper
int	Integer
byte	Byte
short	Short
long	Long
double	Double
float	Float
char	Character

Wrapper Classes – Using MAX_VALUE

The Developer can use constants in the appropriate Wrapper to ascertain the minimum and maximum values permissible in each numeric datatype, they are used like this …

```
byte low = Byte.MIN_VALUE;
byte high = Byte.MAX_VALUE;

System.out.print("Lowest value for a byte is " + low);
System.out.println(" Highest value for a byte is " + high);
```

Lowest value for a byte is -128 Highest value for a byte is 127

The above 'boundaries' of a primitive are necessary when testing within a loop for example.

Wrapper Classes – Declaring Wrappers

A Wrapper datatype can be declared in several ways …

```
Integer count;
Integer count2 = new Integer(0); // It is mandatory to initialise the variable
Integer count3 = new Integer("12");
```

In addition to giving the Developer more scope when constructing a variable, Class wrappers have a variety of methods which allow various operations to be performed on the variable.

The following examples will work for all wrappers not just those illustrated …

Wrapper Classes – Populating Wrapper Classes

If values will need to be inserted into a Wrapper Class (after initialisation) the following 'styles' can be used …

```
count = 12;

count = Integer.decode("13"); // Quotation marks not mandatory

count = Integer.valueOf(12);

count = Integer.parseInt("12");
```

A wrapper class can also be populated using its primitive counterpart …

```
double value = 15.55;
Double count;
count = value;
```

Wrapper Classes – Populating Wrapper Classes

and in addition, a Wrapper can be populated by a variable of the same class …

```
Integer value = 15;
Integer count;

count = value;
```

A primitive datatype of the same type can be populated using the Wrapper class …

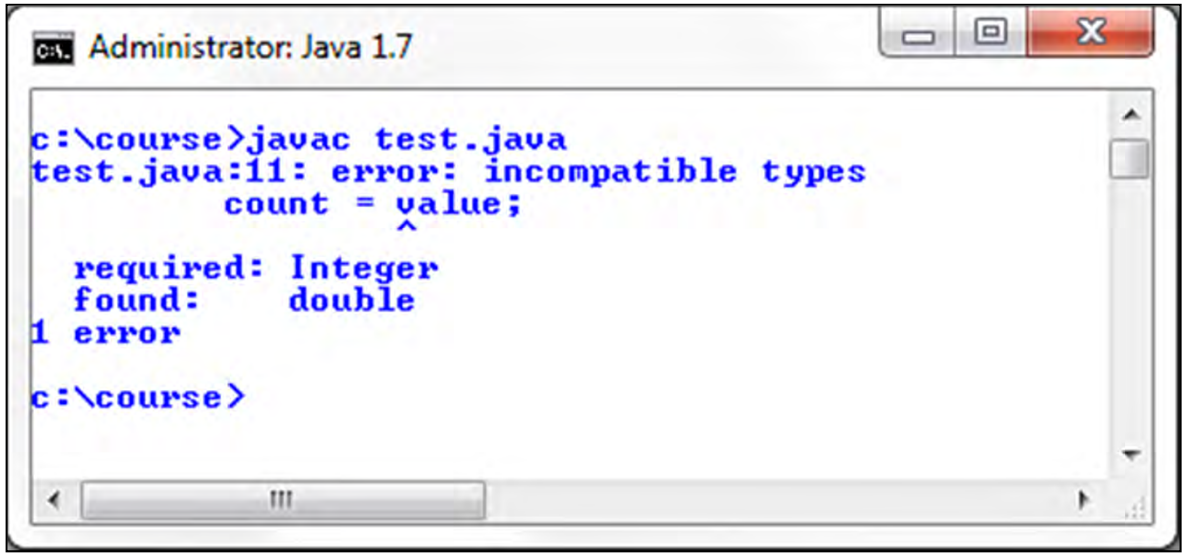
```
Integer Integer_value = new Integer(12);
int int_value = 1;

int_value = Integer_value;
```

Wrapper Classes – Populating Wrapper Classes

Attempting to populate a Wrapper with a value from a 'non-matching' primitive will result in a 'type mismatch' error message …

double value = 15.55;
Integer count;

count = value;

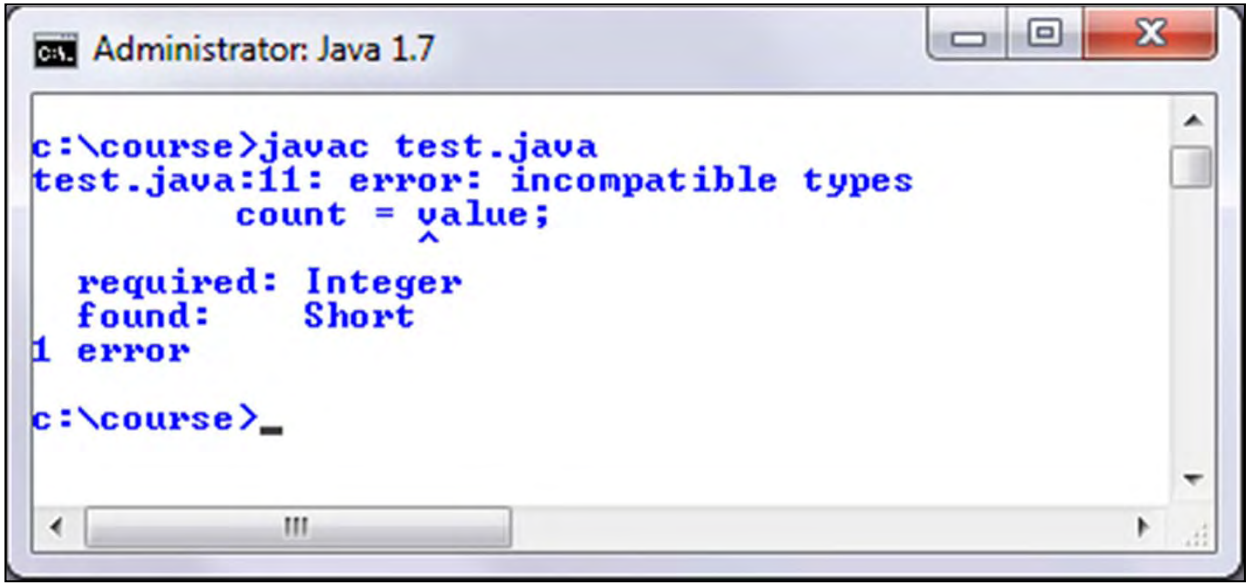


Wrapper Classes – Populating Wrapper Classes

The same happens with non-matching Wrappers …

Short value = 15;
Integer count;

count = value;



Wrapper Classes – Postfix Wrappers

The Postfix technique introduced in the previous section can also be used with the Wrapper classes …

```
Integer Int_value = 0;

Int_value ++;

System.out.println(Int_value);
```

1