

Wrapper Classes

Recap: Primitives vs. References

Primitive	Reference
Most basic data type in Java. Built-in	Insatiable classes made by programmers that often use primitive types.
Store values	Store the address of the value.
Do not have associated methods	Have associated methods

Some objects in Java will only interact with other object.



Wrapper Classes

We can convert primitive types to object types using wrapper classes.

For every every primitive type in Java there is a built-in object type called Wrapper Class which contains or "wraps" primitive data types as an object.



Wrapper Classes

Primitive Type	Corresponding Wrapper Class
boolean	Boolean
char	Character
int	Integer
double	Double



How do Wrapper Classes Help?

Wrapper classes provide a way to wrap primitive values in an object, so that primitives can do activities reserved for objects.

Each wrapper class contains special values (like the minimum and maximum values for the type) and methods that are useful for converting between types.

Wrapper classes provide an assortment of built in utility methods for primitives like converting primitive types to and from string objects or comparing various objects.



Instantiating an Object of a Wrapper Class

```
Integer y = Integer.valueOf(3)
```

```
Double z= Double.valueOf(3.14)
```

Wrapper Classes start with Uppercase, while primitive types use lowercase.

The data type input as an actual parameter must match the Wrapper Class.



Integer and Double Methods

Integer Methods	Use
Integer.MIN_VALUE	Returns the minimum possible int or Integer value
Integer.MAX_VALUE	Returns the maximum possible int or Integer value
Integer.intValue()	Convert Integer value into an int value

Double Methods	Use
Double.doubleValue()	Convert Double value into an double value

Integer Class Methods

`Integer.MIN_VALUE`: The minimum value represented by an `int` or `Integer`
(-2147483648)

`Integer.MAX_VALUE`: The maximum value represented by an `int` or `Integer`
(2147483647)



Integer intValue()

```
Integer y = Integer.valueOf(3)
```

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



Is there an easier way?

The conversion can be done automatically at runtime by the compiler using features called autoboxing and unboxing.

Autoboxing: Includes converting a primitive value into an object of the corresponding wrapper class. It is done automatically by the compiler.



Autoboxing Example

Instead of having to write:

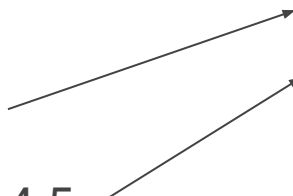
```
Integer y = Integer.valueOf(3);
```

```
Double x = Double.valueOf(4.5);
```

We can write:

```
Integer myInt = 3;
```

```
Double myDouble = 4.5
```



Java automatically converts primitive values to Integer and Double objects if values are declared as Wrapper class objects.

Autoboxing

The Java compiler will also apply autoboxing when a primitive value is passed as parameter to a method that expects an object of the corresponding wrapper class.

```
ArrayList<Integer> myList = new ArrayList<Integer>();
```

```
int primitive = 5;
```

```
myList.add(primitive)
```

Java autoboxes the variable primitive and converts it to an integer object



Unboxing

Converting an object of a wrapper type to its corresponding primitive value is called unboxing.

```
Integer myInt =5;
```

```
int myInt2 = myInt;
```



Autoboxing

The Java compiler will also apply unboxing when a Wrapper value is passed as a parameter to a method that expects a primitive value.

```
Integer wrapperClass = Integer.valueOf(5);
```

```
Rectangle rect = new Rectangle(wrapperClass, 8);
```

```
System.out.println(rect);    =>  w:5 - h: 8
```



Advantages of Autoboxing / Unboxing

Autoboxing and unboxing allow programmers write cleaner code, making it easier to read.

It allow us to use primitive types and wrapper class objects interchangeably without having to perform any casting explicitly.

