type wrapper

Java uses primitive data types such as int, double, float etc. to hold the basic data types for the sake of performance. Despite the performance benefits offered by the primitive data types, there are situations when you will need an object representation of the primitive data type. For example, many data structures in Java operate on objects. So you cannot use primitive data types with those data structures. To handle such type of situations, Java provides **type Wrappers** which provide classes that encapsulate a primitive type within an object.

- **Character :** It encapsulates primitive type char within object.

  ```
  Character (char ch)
  ```

- **Boolean :** It encapsulates primitive type boolean within object.

  ```
  Boolean (boolean boolValue)
  ```

- **Numeric type wrappers :** It is the most commonly used type wrapper.

Byte	Short	Integer	Long	Float	Double

  Above mentioned Classes comes under Numeric type wrapper. These classes encapsulate byte, short, int, long, float, double primitive type.

Autoboxing and Unboxing

- Autoboxing and Unboxing features was added in Java5.
- **Autoboxing** is a process by which primitive type is automatically encapsulated(boxed) into its equivalent type wrapper
- **Auto-Unboxing** is a process by which the value of an object is automatically extracted from a type Wrapper class.

Example of Autoboxing and Unboxing

```
class Test
{
 public static void main(String[] args)
 {
  Integer iob = 100;
  int i = iob;
  System.out.println(i+" "+iob);

  Character cob = 'a';
  char ch = cob;
  System.out.println(cob+" "+ch);
 }
}
```

```
100 100 a a
```

Autoboxing / Unboxing in Expressions

Whenever we use object of Wrapper class in an expression, automatic unboxing and boxing is done by JVM.

```
Integer iOb;
iOb = 100;
++iOb;
```

When we perform increment operation on Integer object, it is first unboxed, then incremented and then again reboxed into Integer type object.

This will happen always, when we will use Wrapper class objects in expressions or conditions etc.

Benefits of Autoboxing / Unboxing

1. Autoboxing / Unboxing lets us use primitive types and Wrapper class objects interchangeably.
2. We don't have to perform Explicit **typecasting**.
3. It helps prevent errors, but may lead to unexpected results sometimes. Hence must be used with care.
4. Auto-unboxing also allows you to mix different types of numeric objects in an expression. When the values are unboxed, the standard type conversions can be applied.

Example:

```
class Test {
public static void main(String args[]) {
Integer i = 35;
Double d = 33.3;
d = d + i;
System.out.println("Value of d is " + d);
}
}
```

```
Value of d is 68.3
```

Note: When the statement **d = d + i;** was executed, i was auto-unboxed into int, d was auto-unboxed into double, addition was performed and then finally, auto-boxing of d was done into Double type Wrapper class.