Wrapper Classes

For every primitive datatype there is corresponding Wrapper class available.

All wrapper classes can be used without import as they are available in java.lang package.

Primitive Data type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Every Wrapper class overrides toString(), hashCode() and equals(Object obj) which is inherited to it from Object class.

Wrapper classes are useful to store primitive data in the objects.

Retrieve primitive value from object.(unboxing)

All the Wrapper classes are final classes.

All wrapper classes except Boolean and Character are the subclasses to Number class.

1.	**Why do we need Wrapper Classes?** To make program 100% object oriented. If you are using primitive it will not be 100% object oriented so using Wrapper classes primitive variables equivalent object can be created and used. E.g. Java Collection Frameworks allows only objects.If you want to store an primitive data into Collection such as ArrayList, convert in appropriate Wrapper and then store in ArrayList.
2.	**Which Wrapper class has only one constructor?** Character
3.	**Tell something about varieties of constructors in different wrapper classes** All Wrapper classes except Character has 2 constructors: 1. Taking 1 corresponding primitive type literal/ variable as argument. 2. Taking 1 String object as an argument. Boolean wboo = new Boolean("false"); Boolean yboo=new Boolean(false); Byte wbyte = new Byte("2"); Byte ybyte=new Byte(2);

	Short wshort = new Short("4");
	Short yshort = new Short(4);
	Integer wint = new Integer("16");
	Integer yint = new Integer(16);
	Long wlong = new Long("123");
	Long ylong = new Long(123);
	Float wfloat = new Float("12.34f");
	Float yfloat = new Float(12.34f);
	Double wdouble = new Double("12.56d");
	Double wdouble = new Double(12.56d);
	Character c1 = new Character('c');
	
	`boolean boo = false;`
	`Boolean wboo = new Boolean(boo);`
	`byte b = 2;`
	`Byte wbyte = new Byte(b);`
	`short s = 4;`
	`Short wshort = new Short(s);`
	`int i = 16;`
	`Integer wint = new Integer(i);`
	`long l = 123;`
	`Long wlong = new Long(l);`
	`float f = 12.34f;`
	`Float wfloat = new Float(f);`
	`double d = 12.56d;`
	`Double wdouble = new Double(d)`

4.	**Apart from constructor, Is there any other way to create corresponding Wrapper object ?** In all Wrapper classes except Character there are 2 overloaded static valueOf() method. 1. Taking 1 corresponding primitive type literal or variable as argument. 2. Taking 1 String object as an argument.
5.	**How many toString() are available in Wrapper classes?** There are 2 toString() are available in every Wrapper classes: 1. Inherited from Object class toString() i.e. no-arg 2. static toString() with 1 argument.
6.	**In valueOf() if String provided contains an alphabet then will it work ?** NumberFormatException.

7.	**How to convert String into primitive?** Using parseXXX() except for Character.
8.	**In which version of JDK, autoboxing and unboxing concept has been introduced?** JDK 1.5
9.	**What happens if we try to assign int variable to an Integer object straightaway in case of JDK1.4 or earlier version?** It will give compile time error.
10.	**In if condition if Boolean object is provided will it work?** If it is for JDK1.5 or later version then it will work fine. For JDK1.4 or earlier version it will result in compile time error.
11.	**Tell something about Autoboxing.** Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on. E.g. Character ch = 'a';
12.	**Tell something about Unboxing.** Unboxing is the automatic conversion that the Java compiler makes between the object wrapper classes and their corresponding primitive types. For example, converting an Integer to an int, a Double to a double, and so on. E.g. Character ch = ‘a’; char c = ch;
13.	**Explain about vararg in java.** JDK1.5 or later version supports vararg. It allows the method to accept zero or multiple arguments. Syntax for vararg: return_type method_name(data_type... variableName){} varargs uses ellipsis i.e. three dots after the data type. There can be only one variable argument in the method. Variable argument (varargs) must be the last argument.
14.	**When there is a situation to choose between widening and boxing and vararg specify the preference?** 1. Widening 2. Boxing 3. Vararg 1. Widening is preferred over boxing:

	When there is a situation to choose between widening and boxing, widening takes the preference. 2. Widening is preferred over vararg: When there is a situation to choose between widening and varargs, widening takes the preference 3. Boxing is preferred over vararg: When there is a situation to choose between boxing and varargs, boxing takes the preference