Library Classes & Wrapper Classes

For better understanding:

Before we get into Library Classes & Wrapper Classes, it's important to know what is a primitive and composite data types.

Primitive Data Type: These are **fundamental built-in data types** of **fixed sizes**. **Ex:** int, long, float

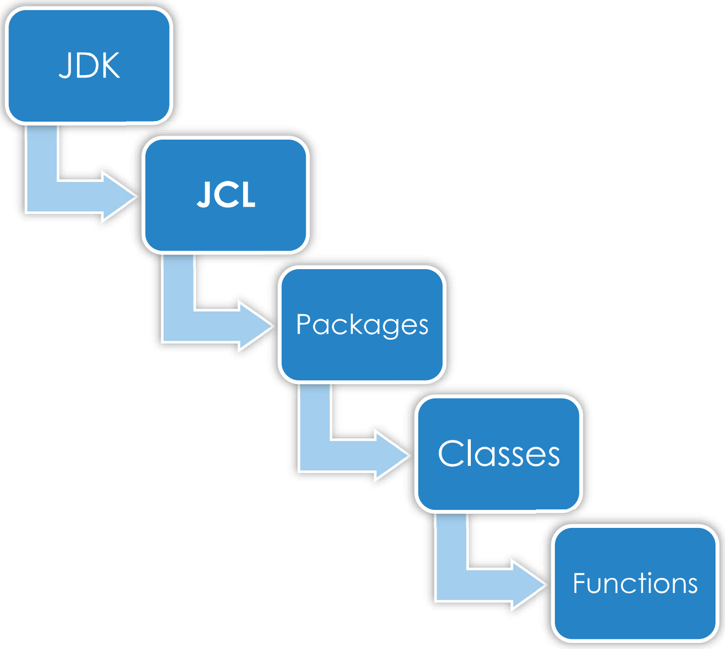
- **Composite/Reference/User-Defined Data Type:** These are data types **created by the user**. The availability of these data types depends upon their **scope and sizes** depend upon their **constituent members**. **Ex:** array, class, object

Primitive data type	Composite data type
These are built in data types	Created by user
The sizes of these objects are fixed	The sizes of these data types depend upon their constituent members
These data types are available in all parts of a java program	The availability of these data types depends upon their scope

Difference between primitive and composite data type.

Library Classes

JDK (Java Development Kit) V1.5 and above contains Java Class Library (JCL) which contains various packages. Each package contains various classes containing different built-in functions.



Ex: *java.lang, java.math*

Wrapper Class

Wrapper Classes are a part of java.lang (A Library Class Package). Wrapper classes **contain primitive data values in terms of objects**/ Wrapper Class **wraps a primitive data type to an object**. There are 8 wrapper classes in Java. **Ex:** Integer, Byte, Double

(**NOTE:** Wrapper Classes always start with an uppercase letter
Ex: Integer, Boolean, Float)

Need for Wrapper Classes

- To store primitive values in the objects
- To convert a string data into other primitive types and vice-versa

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

Wrapper Classes and their primitive types

Functions/Methods of Wrapper Classes

Conversion from String to Primitive types

For converting String to any primitive data type, Wrapper Class functions can be used. For any primitive data Wrapper Class, the **parse**<prm data type>(<String arg>) (or) **valueOf**(<String arg>) functions can be used.

Eg: int i=Integer.parseInt(s); int j=Integer.valueOf(s);

For better understanding:

```
<prm data type var>=<prm data type Wrapper Class>.parse<prm data type name>(<String arg>);
<prm data type var>=<prm data type wrapper class>.valueOf(<String arg>);

//Examples:
int a=Integer.parseInt("238");
doubleb=Double.parseDouble("23.45");
int c=Integer.valueOf("37");
float d=Float.valueOf("42.87");
```

Examples of each <> (In the above syntax):

prm data type: int a | double b
prm data type name: Int | Long | Double
prm data wrapper class: Integer | Double
String arg: "38.743" | "1874293856"

Conversion from primitive type data to String

For converting a primitive type data to a String, the **toString()** Wrapper Class function can be used.

Ex: Integer.toString() | Double.toString()

```
<String var>=<Wrapper Class>.toString(<prm data arg>);
String cnv=Integer.toString(38);
String dbl=Double.toString(94.53);
```

Boxing, Unboxing & Autoboxing

Boxing

Conversion of primitive type data **to an object**.

Syntax with example:

```
<wrapper class> <object name>=new <wrapper class>(<prm type arg>);
int a=239;
Integer x=new Integer(a);
```

Unboxing

Conversion of an object **to primitive type data**.

Syntax with example:

```
<int var>=<wrapper class obj>
int b=x;
```

Autoboxing

Boxing is the mechanism and autoboxing is the feature of the compiler which generates the boxing code.

Syntax with example:

```
<wrapper class> <object name>=new <wrapper class>(<prm type arg>);
int a=239;
Integer x=new Integer(a);
```

Character

Character is defined as a letter, a digit or any special symbol/UNICODE enclosed within single quotes. **Ex:** ‘@’, ‘s’, ‘5’

Assigning a character

A Character is declared under char data type.

Syntax with example:

```
char <var name>=’<char literal>’;
char ch=’a’;
```

Input a character

A Character is declared under char data type.

Syntax with example:

```
<char var>=<Scanner obj>.next().charAt(0);
ch=sc.next().charAt(0);
```

Character Functions

Character.isLetter() (boolean)

This function is used to check if a given argument is a letter or not.

Syntax with example:

```
<boolean var>=Character.isLetter(<char arg>);
boolean chk=Character.is(‘A’); //true
```

Character.isDigit() (boolean)

This function is used to check if a given argument is a digit or not.

Syntax with example:

```
<boolean var>=Character.isDigit(<char arg>);
boolean chk=Character.is(‘7’); //true
```

Character.isLetterOrDigit() (boolean)

This function is used to check if a given argument is either a letter or a digit or none of these.

Syntax with example:

```
<boolean var>=Character.is(<char arg>);
boolean chk=Character.is(‘A’); //true
```

Character.isWhitespace() (boolean)

This function is used to check if a given argument is a blank/gap/space or not.

Syntax with example:

```
<boolean var>=Character.is(<char arg>);
boolean chk=Character.is(‘A’); //false
```

Character.isUpperCase() (boolean)

This function is used to check if a given argument is an uppercase letter or not.

Syntax with example:

```
<boolean var>=Character.is(<char arg>);
boolean chk=Character.is(‘A’); //true
```

Character.isLowerCase() (boolean)

This function is used to check if a given argument is a or not.

Syntax with example:

```
<boolean var>=Character.is(<char arg>);
boolean chk=Character.is(‘A’); //false
```

Character.toUpperCase() (char)

This function is used to convert/returns a given argument/character/letter to/in uppercase character/letter.

Syntax with example:

```
<char var>=Character.toUpperCase(<char arg>);
char uc=Character.toUpperCase(‘a’); //A
```

Character.toLowerCase() (char)

This function is used to convert/returns a given argument/character/letter to/in lowercase character/letter.

Syntax with example:

```
<char var>=Character.toLowerCase(<char arg>);
char lc=Character.toLowerCase('A'); //a
```