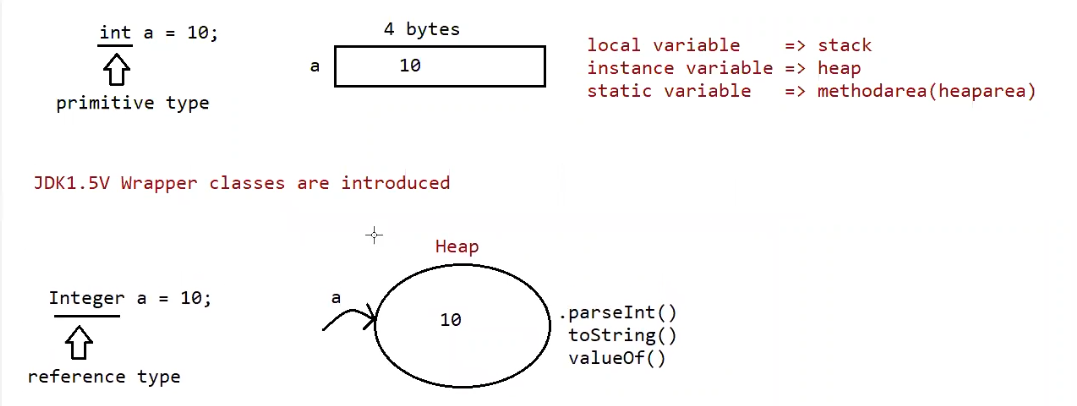
Wrapper classes:



Note: we cannot use instance variables inside static method (or) block, because instance variables are associated with the object creation (object needs to created manually by the user to allocate memory to the instance variables and assign the values to them.)

When we create a variable of primitive data type, memory for it is assigned based on whether it is local, instance, static variable.

But when we create a variable of wrapper class, the memory is not allocated as primitive data type, instead a object will be created and data is stored in the object automatically by the jvm.

So Wrapper classes can be used in the static blocks and methods also.

The reason for using wrapper classes is it creates object, and with that reference of the object we can call some methods eg: .parseInt(), .toString(), .valueOf() (to check all the supported methods for Integer use command javap java.lang.Integer .For remaining wrapper classes use their names along with the above command)

**Java Utility Classes**

A utility class in Java is a **class that provides static methods that are accessible for use across an application**. The static methods in utility classes are used for performing common routines in our application.

**Utility classes cannot be instantiated** and are sometimes stateless without static variables. We declare a utility class as [*final*](https://www.baeldung.com/java-final)*,* and all its methods must be *static*.

Since we don’t want our utility classes to be instantiated, a private constructor is introduced. Having a **private constructor means that Java won’t create a default constructor for our utility class**. The constructor can be empty.

Java has utility classes such as *java.util.Arrays*, *java.lang.Math*,  *java.util.Scanner*,  *java.util.Collections,* etc.

**How to Create a Java Utility Class**

Creating a utility class is not so different from how we create a helper class. A few things are done a little differently when creating a utility class.

To create a utility class, **we use a *public* access modifier and also declare the class as *final***. The *final* keyword used when creating utility classes means that the class would remain unchangeable. It cannot be inherited or instantiated.

Eg: User\_Defined\_Utility\_Class

Another rule to observe is that all methods of a utility class are *static,* with a *public* access modifier.

Since we have only static methods within utility classes, these methods can only be accessed via the class name:

Helper classes:

A helper class provides functionalities necessary for the overall running of a Java program. Helper classes**contain methods used by other classes to perform repetitive tasks, which aren’t the core purpose of an application**

A helper class can be instantiated and may contain [instance variables](https://www.baeldung.com/java-static-instance-initializer-blocks), instance, and [static](https://www.baeldung.com/java-static) methods.

Multiple instances of a helper class can exist in our application. When different classes have common functionalities, we can group these functionalities together to form a helper class that’s accessible across certain classes in our application.

**How to Create a Java Helper Class**

To create a helper class, we use a default access modifier for our class name. The default access modifier ensures that only classes within the same package have access to this class, its methods, and variables

Eg: Helper\_Class

After defining the class, we can add as many related instance and static methods as needed.

Helper classes can have [instance methods](https://www.baeldung.com/java-class-methods-vs-instance-methods) because they can be instantiated.

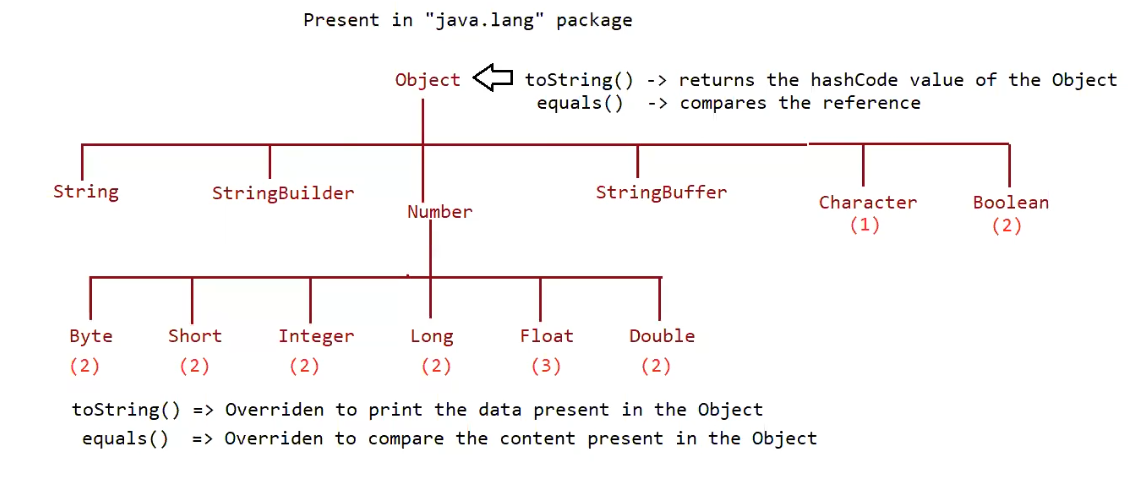
For every class parent happens to be Object, but for wrapper classes the parent happens to be Number class, and Object is the parent of Number class

toString() is present in object class. (to string() returns the hashcode value of the object)

Purpose of wrapper classes:

A Wrapper class in Java is a class that wraps around a primitive data type and converts it into an object. Wrapper classes provide a way to treat primitive data types, such as int or double, as objects.

Purpose of wrapper classes is to define several utility functions which are required for the primitives



* Almost all the wrapper classes have 2 constructors eg: Byte, Short, Integer, Long, Float, Double.
* But Character and Boolean have only 1 constructor.
* Almost for all the user defined and predefined classes the parent is Object class.
* But for Byte, Short, Integer, Long, Float, Double wrapper classes the parent is Number class.
* Object class contains the toString() , equals() that have above functionality.
* The methods can be overridden by classes and they may have some other functionality.

Example equals() of String class compares the content not the reference.

* The toString() method of wrapper classes are overridden and have the above functionality.

Constructors (check using the command javap java.lang.Integer, and for remaining wrapper classes use the above command just the name after lang.Wrapper\_Class\_Name )

1. One taking primitive type

eg: public java.lang.Integer(int);

Integer a = new Integer (10);

1. One taking String type

Eg: public java.lang.Integer(java.lang.String) throws java.lang.NumberFormatException;

Integer a1 = new Integer (“10”);

Note: If String argument is not properly defined then it would result in RunTimeException called “NumberFormatException”

Eg: Integer I = new Integer(“ten”) // RuntimeException: NumberFormatException.

Wrapper class and its associated constructor

Note: check every class constructors by javap java.lang.Wrapperclass\_Name

Byte => byte and String

public java.lang.Byte(byte);

public java.lang.Byte(java.lang.String) throws java.lang.NumberFormatException;

Short => short and String

public java.lang.Short(short);

public java.lang.Short(java.lang.String) throws java.lang.NumberFormatException;

Integer => int and String

Eg: Integer\_Constructors // go through the code.

Long => long and String

public java.lang.Long(long);

public java.lang.Long(java.lang.String) throws java.lang.NumberFormatException;

Float => float, double and String

Eg: Float\_Constructors // go through the code

public java.lang.Float(float);

public java.lang.Float(double);

public java.lang.Float(java.lang.String) throws java.lang.NumberFormatException;

Double => double and String

public java.lang.Double(double);

public java.lang.Double(java.lang.String) throws java.lang.NumberFormatException;

Character => character

Eg: Character\_Constructors

public java.lang.String toString();

Boolean => boolean and String

public java.lang.String toString();

Eg: Boolean\_Constructors

Note:

Incase of wrapper class, toString() is overridden to print the data.

Incase of wrapper class, equals() is overridden to check the content.

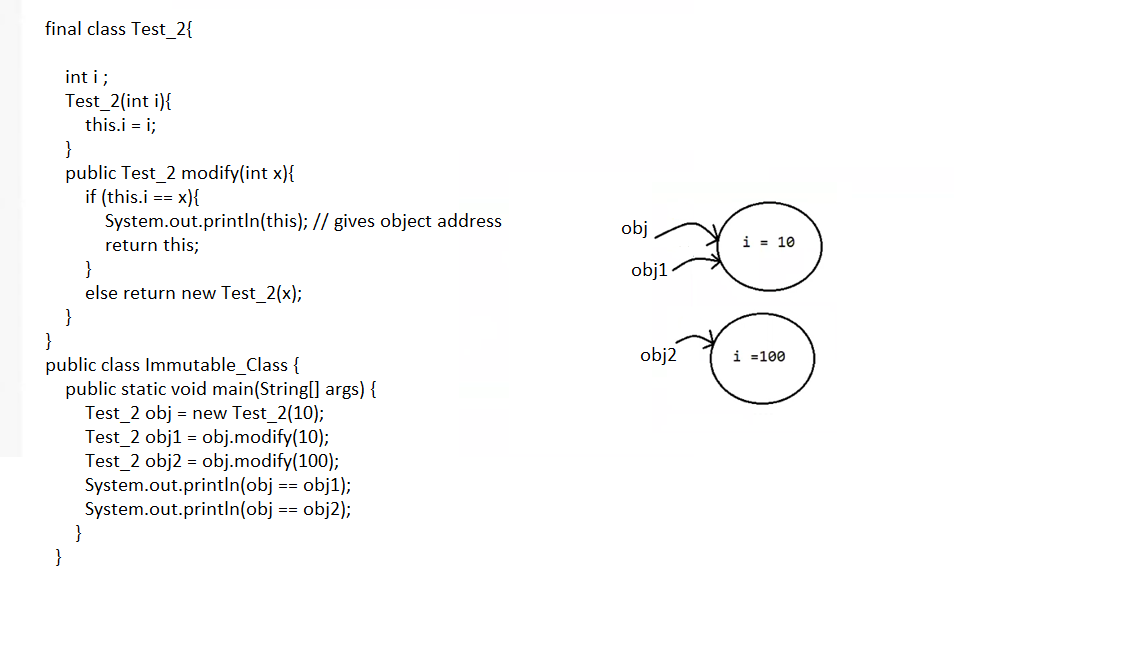
Just like String class, Wrapper classes are also treated as “immutable class”.

Eg: Interger\_Equals\_Method

// go through the code

Can we make our user defined class immutable?

Yes, shown as below.



## [Creating an Immutable Class in Java](https://www.digitalocean.com/community/tutorials/how-to-create-immutable-class-in-java#creating-an-immutable-class-in-java):

1. Declare the class as final so it can’t be extended.
2. Make all of the fields private so that direct access is not allowed.
3. Don’t provide setter methods for variables.
4. Make all mutable fields final so that a field’s value can be assigned only once.
5. Initialize all fields using a [constructor](https://www.digitalocean.com/community/tutorials/constructor-in-java) method performing deep copy.
6. Perform [cloning](https://www.digitalocean.com/community/tutorials/java-clone-object-cloning-java) of objects in the getter methods to return a copy rather than returning the actual object reference.

## Eg: User\_Defined\_Immutable\_Class

Wrapper classes and string classes are immutable (immutable means if you try to make a change to the object with that change a new object is created)

toString() => prints the content of the object

equals() => compares the data present in the object

// both the methods works same in String, wrapper class.

Note: Methods which are static are said to be helper methods (or) utility methods.

note: if you are trying to keep the data in the string and if that data is not of integer type, then it would result in the Number Format Exception.

Wrapper classes utility methods:

1. valueOf() method
2. XXXvalue() method
3. parseXxx() method
4. toString() method

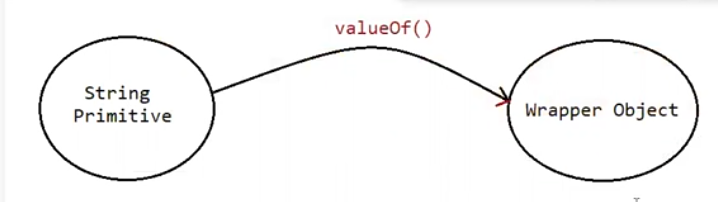
( javap java.lang.Wrapper\_ClassName will give the utility methods .)

public static Wrapper valueOf (String data, int base (or) radix) throws java.lang.NumberFormatException;

public static Wrapper valueOf (String data) throws java.lang.NumberFormatException;

public static Wrapper valueOf (int);

valueOf() method:



To create a wrapper object from primitive type (or) String type we use valueOf() method

It is an alternative to constructor of Wrapper class, not suggestable to use.

Every wrapper class, except character class contain static valueOf() method to create wrapper object.

Eg: ValueOf\_Method

// go through the code

Eg: ValueOf\_Method\_Eg2

// go through the code

public static valueOf(String s , int radix)

|=> binary: 2(0,1)

<https://www.youtube.com/watch?v=sXxwr66Y79Y>

Note: Except binary, working procedure of all the remaining number systems are same.

Not only this mentioned number systems java supports 2,3,4,5,6,7,8,9,10,11,12,13…… 36 number systems.

| => octal: 8(0-7)

https://www.youtube.com/watch?v=XHXNUHMQ1IM

|=> decimal: 10(0-9)

https://www.youtube.com/watch?v=G4doTGQPRPw

|=> hexadecimal: 16(0-9, a, b, c, d, e, f)

// a=10, b=11.. f = 15

|=>base: 36(0-9, a-z)

// a=10, b=11… z= 35

Note: the minimum and maximum base we can give is

Eg: Min\_Max\_Radix // go through the code

Eg: ValueOf\_Eg3

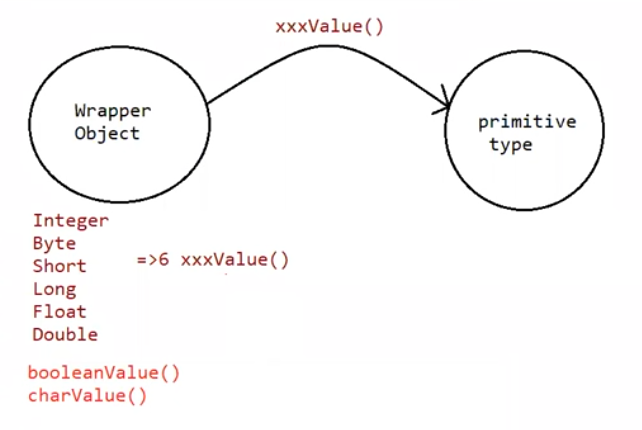
// check valueOf() method For Boolean , Character takes which types of arguments

javap java.lang.Boolean

javap java.lang.Character

// go through the code

2. xxxValue()



We can use xxxValue() to get primitive type for the given wrapper object.

These methods are a part of every Number type Object.

(Byte, Short, Integer, Long, Float, Double) all these classes have these 6 methods which are

Methods:

public byte byteValue();

public short shortValue();

public int intValue();

public long longValue();

public float floatValue();

public double doubleValue();

Eg: Integer\_Value\_Method

// go through the code

Character class has charValue() to get character primitive for given character object.

public char charValue()

Eg:Character\_Value\_Method

Boolean class has booleanValue() to get boolean primitive for given boolean object.

public boolean booleanValue()

Eg: Boolean\_Value\_Method

In total xxxValue() are 36 in number

xxxValue() => convert the wrapper object => primitive

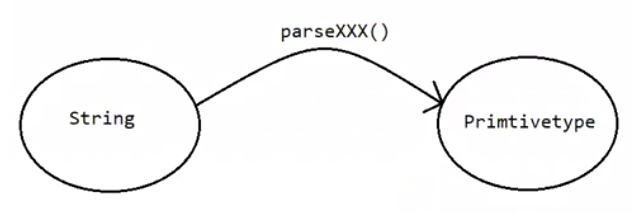
3. parseXXX() :

We use parseXXX() to convert String Object into primitive type

Form-1:

public static primitive parsexxx (String s)

Every wrapper class except Character class has parseXXX() to convert string to primitive



Eg: Parse\_Method\_Eg1

Usage of wrapper class in real time coding:

Eg: Wrapper\_Class\_Parse\_Method\_RealTime\_Usage

Note: The command line arguments are taken as String in java program, because the string input from user can be converted to all types of primitives except char.

You can pass character in command line, but it is treated as String only. there is no parse method to convert String to character primitive.

Form-2:

public static primitive parsexxx (String s, int radix)

radix -> range is from 2 to 36.

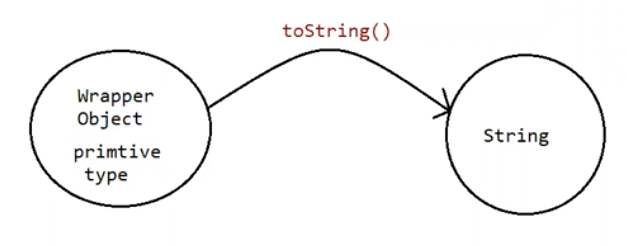
Every Integral type wrapper class (Byte, Short, Integer, Long) contains the following parseXXX(String s, int radix) to convert specified radix string to primitive

// check javap java.lang.Wrapperclass

Eg: Parse\_Method\_Radix

// go through the code

toString() :



To convert the wrapper object (or) primitive to string

Every wrapper class contain toString() method.

From:

public String toString()

1. Every wrapper class (including the Character class) contain the above toString(), to convert wrapper object to String.
2. It is the overriding version of Object class toString() method.
3. Whenever we are trying to print wrapper object reference toString() method is automatically called.

Eg: Integer a = Integer.valueOf(“10”);

System.out.println(a) // internally it calls toString() and prints the data.

Form2:

public static String toString(primitive)

1. Every wrapper class contain a static toString() method to convert primitive to string.

eg: String s = Integer.toString(10);

Eg: Primitive\_ToString

Form 3:

Integer and Long classes contains the following static toString() to convert the primitive to specified radix String form.

public static String toString (primitive p, int radix)

Eg: Primitive\_Radix\_ToString

Form 4:

Integer and Long classes contains the following toXxxString() methods.

public static String toBinaryString (primitive p)

public static String toOctalString (primitive p)

public static String toHexString(primitive p)

Eg: ToString\_For\_Binary\_Octal\_Hex

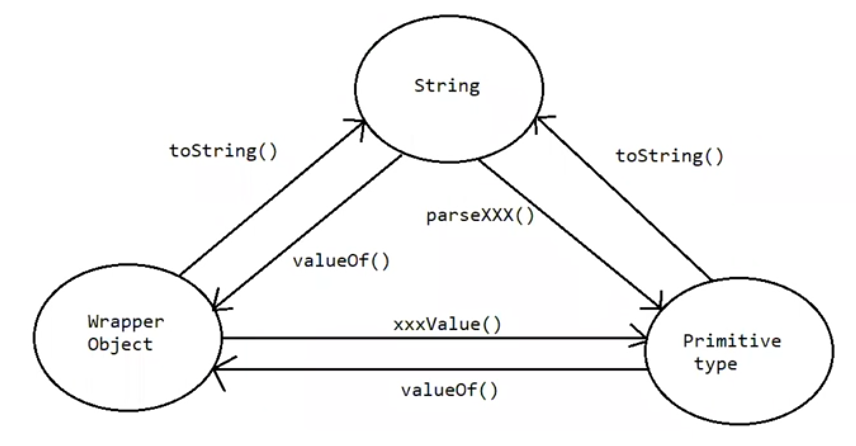
https://www.youtube.com/shorts/3-kHd\_CWAM0

valueOf() -> String / primitive to wrapper object.

XXXValue() -> wrapper to primitive

parseXXX() -> String to primitive

toString() ,toHexString() , toBinaryString() , toOctalString() -> primitive to String.



Note :

String class

public static String valueOf(boolean);

public static String valueOf(char);

public static String valueOf(int);

public static String valueOf(long);

public static String valueOf(float);

public static String valueOf(double);

String data = String.valueOf(‘a’); // static factory methods

Here we are performing operations on String and output is also string type we call them as factory methods.

String data = “pavan”.toUpperCase(); // instance factory methods

Here we are performing operations on StringObject and output is also string type we call them also factory methods.