**StringBuffer**

* StringBuffer sb = new StringBuffer();

Create a empty string Buffer object with default initial capacity (16) once stringbuffer reach at max capacity a new StringBuffer object will be created with new capacity ==

New Capacity = (current capacity +1 ) \*2

* StringBuffer sb = new StringBuffer(int initial capacity);

Create a empty string buffer object with initial capacity

* StringBuffer sb = new StringBuffer(String s);

Creats a equivalent StringBuffer for the given string with

capacity = s.length() + 16

Important method of StringBuffer :-

1. Public int length();
2. Public int capacity();
3. Public char charAt(int index);
4. Public void setCharAt(int index, char ch); // to replace the character located a specified index with provided character
5. Public StringBuffer append(String s); // this method we can apply for all data types because it is overloading method
6. Public StringBuffer insert (int index, String s ) // (int index, int i ) (int index, double d ) (int index, char ch ) (int index, Boolean b )
7. Public StringBuffer delete (int begin, int end) // to delete character located from begin index to end-1 index
8. Public StringBuffer deleteCharAt(int index) // to delete the character located at specified index
9. Public StringBuffer reverse() // to reverse the string order

When we go for string and stringbuffer and string Builder

String :- if the contain is fixed and won’t change frequently then we should go for string

StrigBuffer:- If the contained is not fixed and keep on changing but thread safe is required then we should go for string buffer

Stringbuider:- If the caintain is not fixed keep on changing but thread safe is not required then we should go for string builder

**Method chaining**

For most of the method string , stringBuffer and StringBuilder return types are same type hence after apply a method a result we can call another method which forms method chaining

Sb.m1().m2().m3().m4()….

In method chaining method call will be executed from left to right

**Wrapper classes :-**

1. The main objective wrapper classes are to wrap primitive into object form so that we can handle primitive just like objects
2. To define several utility method which are required for primitive

Constructor :-

1. Integer I = new integer (10);
2. Integer I = new integer(“10”);

Almost all wrapper classes contain two constructor one can take corresponding primitive as argument and other can take string as argument

* If string argument not representing a number then we will get runtime exception saying number format exception

Eg. Integer I = new Integer (“ten”);

* Float class contain 3 constructor with float , double and string arguments

Eg. Float f = new Float(10.0f);

Float f = new Float(“10.0f”);

new Float(10.0);

new Float(“10.0”);

* character class contain only on constructor which can take care argument

eg. Character ch = new Character (‘a’)

* Boolean class contain two constructor Boolean and string

If we pass Boolean primitive as argument the only allowed value are true or false where case in important and content is also requirement

If the contain is case insensitive string of true then it is treated as true otherwise it is always false

Eg.:-

Boolean b = new Boolean(“true”); 🡪true

Boolean b = new Boolean(“TrUe”); 🡪true

Boolean b = new Boolean(“TRUE”); 🡪true

Boolean b = new Boolean(“mallika”); 🡪false

Boolean b = new Boolean(“tmalaika”); 🡪false

|  |  |
| --- | --- |
| Wrapper class :- | Corresponding constructor argument:- |
| Byte | byte or String |
| Short | short or String |
| Integer | int or String |
| Long | long or String |
| Float | float or String or double |
| Double | String or float |
| Character | char |
| Boolean | boolean or String |
|  |  |
|  |  |
|  |  |

To all wrapper classes toString() method is overridden to return contain directly

.equals() method is overridden for contain comparison

Utility methods:-

* valueOf()
* xxxValue()
* parseXxx()
* toString()

1. valueOf() method :- we can use value of methods to create wrapper object for the given primitive or string

* form 1 :- public static wrapper valueOf(String s)

every wrapper class contain except character class a static value of method to create wrapper object for the given string

eg.

Integer I= Integer.valueOf(“10”);

Double D = Double.valueof(“10.5”);

Boolean b = Boolean.valueOf(“true”);

**Hash table**

1. if two object are equal by .equals method then there hashCode() must be equal that us to equivalent should have same hashCode()

r1.equals(r1) // true then r1.hashCode() == r2.hashCode()

Object class .equals() and HashCode() follows above contract hence whenever we are overriding that equals method compulsory we should override hashCode() method to satisfy above contract (that is two equivalent object have same hash Code

If two object are not equal by .equals method then there is no restriction on hash code maybe equals or not be equal like

If hashCode of two object are equals then we can’t conclude any think about .equals method it may return true or false

If hashCode of two object are not are not equal then this object are always not equal by .equals method