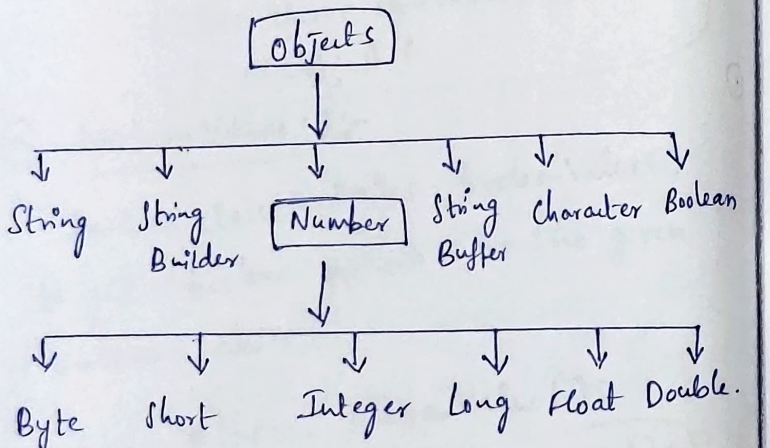
(31) Wrapper Class Continuation

↳ 22nd Nov live class
↳ Nitin Sir

① Need of wrapper class :-

To store primitive data in the form of object so that we can use utility Methods



→ type javap java.lang.Integer in CMD
↳ we see valueOf() Method in Integer wrapper class.
↳ these Methods are also available in all wrapper classes.
→ Which Ever Method is static those Method we call as as utility Methods (or) helper Methods.

→ ParseInt() Method
↳ it is in Integer so parseInt()
↳ in byte it is ParseByte().

→ ~~doing~~ ~~wrapper~~
→ In Every wrapper class we have Equivalent primitive type Methods which gives primitive value.

① Public byte byteValue();
② Public short shortValue();
③ Public Integer IntegerValue();
④ Public long longValue();
⑤ Public Float floatValue();
⑥ Public Double DoubleValue();

→ toString() is Method present in object Class.
→ toString() method is overloaded in wrapper Class.

② Wrapper Class utility Methods

① ValueOf() Method :- To Convert Primitive type (or) string type to wrapper object.

② xxxValue() :- To Convert wrapper object to Primitive type

③ Parsexxx() :- To Convert string type to Primitive type.

④ toString() :- Primitive type (or) wrapper type to string type.

① ValueOf() Method :-

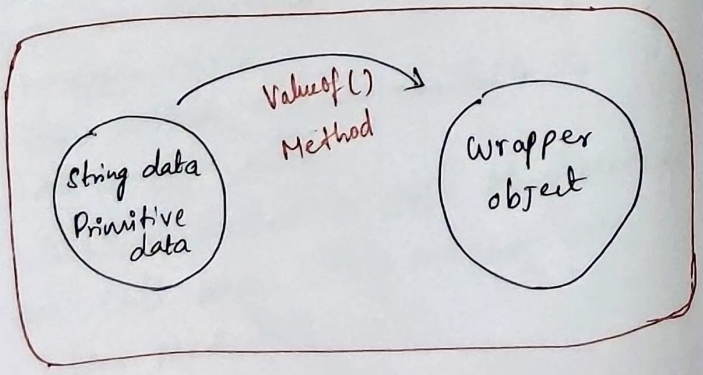
ValueOf() Method is used to Convert string and primitive type data to wrapper type.

(or)

→ To Create a wrapper object from primitive type (or) string we use ValueOf().

→ it is alternative to Constructor of wrapper class, not suggestable to use.

→



→ there are 3 from of ValueOf() Method Prevent.

ⓐ Public Static java.lang.Integer ValueOf(int);

ⓑ Public static java.lang.Integer ValueOf(java.lang.String);

ⓒ Public static java.lang.Integer ValueOf(java.lang.String, int);

① Primitive type data to wrapper type.

Case-0 :- Integer and string as input

ⓐ Integer as input

Public Static java.lang.Integer ValueOf(int);

Eg:

```
Integer i1 = Integer.ValueOf(10);
S.O.P(i1); // 10
```

→ here 10 is Integer (Primitive type) Converted to wrapper type.

1.2 String data to wrapper type

↳ Public Static java.lang.Integer ValueOf(java.lang.String);

```
Integer i2 = Integer.ValueOf("10");
S.O.P(i2); // 10
```

→ here 10 is String Converter to wrapper type.

(1.3) When String type data is passed into valueof() Method instead of Integer type then it throughs Exception.

→ it gives NumberFormatException because here ten is not Integer type.

"ten" → not allowed

"10" → allowed.

↳ here 10 is Integer type given as String to valueof() Method

→ "ten" → here "ten" is String type given as String to valueof() Method it throughs NumberFormatException.

Eg:

```
Integer i3 = Integer.valueOf(i3);
S.o.p(i3);
```

Case (2):- here input is radix (or) base

Number System

(1) base - 2 → binary

(2) base - 8 → octal

(3) base - 10 → decimal

(4) base - 16 → Hexa-decimal

→ Public static java.lang.Integer valueOf(java.lang.String, Radix);

(1.4) 100 base 2 into wrapper type:-

Eg:

```
Integer i4 = Integer.valueOf("100", 2);
S.o.p(i4); // 4
```

⇒ 100 base 2

: 100 → $1\times2^2 + 0\times2^1 + 0\times2^0$

→ 4+0+0 ⇒ 4

(1.5) Giving base which is unknown

↳ it gives NumberFormatException.

Eg:

```
Integer i5 = Integer.valueOf("100", 37);
S.o.p(i5); // NumberFormatException:
           radix 37 greater than
           Character.MAX_RADIX
```

→ Character.MAX_RADIX and Character.MIN_RADIX
→ type in cmd → javap java.lang.character

↳ we find MAX_RADIX and MIN_RADIX in character wrapper class.

(i) Public Static final int MAX_RADIX;
(ii) Public Static final int MIN_RADIX;

→ here Variables are static and final so it has to declared there itself.

⇒ finding MAX_RADIX and MIN_RADIX

```
S.o.p(Character.MAX_RADIX); // 36
S.o.p(Character.MIN_RADIX); // 2
```

→ Number System Configured for JVM.
→ Possibility of number we can give is 2 to 36.
→ Mostly we use Radix from 2 to 16.

② Character type to wrapper type :-

→ In Character wrapper class ValueOf() Method takes only character as input.
→ type in CMD → javap java.lang.Character
↳ see ValueOf() Method
↳ Public static java.lang.Character ValueOf(char)

Eg:
```
Character c = Character.ValueOf('a');
System.out.println(c); // a
```

→ char type to Character Type (Primitive type) (wrapper Type)

③ boolean Type to wrapper type :-

→ In Boolean wrapper class ValueOf() Method takes boolean and String as input.
→ Type in CMD → javap java.lang.Boolean
↳ see ValueOf() Method.
↳ it takes boolean and string as input.

Eg: ③.1 boolean as input.

```
Boolean b1 = Boolean.ValueOf(true);
S.o.p(b1);
```

Eg: (3.1) boolean & Input for valueOf() Method

↳ boolean Type to Boolean Type
(Primitive type) (Wrapper type)

```
Boolean b1 = Boolean.valueOf(true);
S.o.p(b1); //true
```

↳ here 'true' is boolean type Converted to Boolean type (wrapper class).

(3.2) String as input for Boolean valueOf() Method

→ String Type to Boolean Type

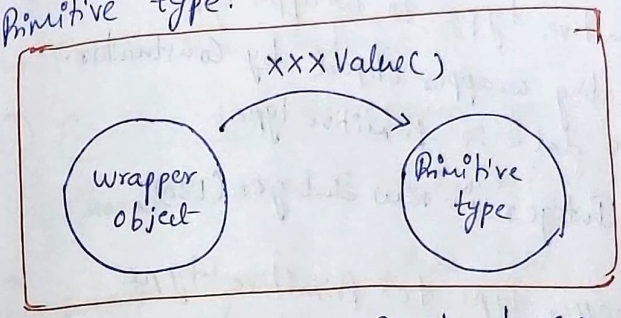
```
Boolean b2 = Boolean.valueOf("aslish");
S.o.p(b2); //false
```

(4) Double type to wrappe type

```
Double d1 = Double.valueOf(10.5);
S.o.p(d1); //10.5
```

→ here, valueOf() Method accepts String or double
(i) Public Static java.lang.Double valueOf(java.lang.String)
(ii) Public Static java.lang.Double valueOf(Double);

(2) xxxValue() :- To Convert to wrapper object to Primitive type.



→ wrapper type → Primitive type.

→ for example we Can Convert Integer type to byte, short, int, long, double, float.

→ we have 6 wrapper classes in Number type

(i) Public byte byteValue();
(ii) Public short shortValue();
(iii) Public int intValue();
(iv) Public long longValue();
(v) Public float floatValue();
(vi) Public double doubleValue();

→ we Cannot Convert Boolean and Character to byte, short, int, float, double, long.

→ xxxValue() Method not applicable for Character and Boolean.

→ We can Convert Character and Boolean to its primitive types Char and boolean.

① Primitive type to wrapper type

→ Creating wrapper object by Constructor.

→ here data is Primitive type:

```
Integer i = new Integer(130);
```

(00) //wrapper type to Primitive type

```
S.o.p(i.byteValue); //-126
S.o.p(i.shortValue); //130
S.o.p(i.intValue); //130
S.o.p(i.longValue); //130
S.o.p(i.floatValue); //130.0
S.o.p(i.doubleValue); //130.0
```

② Character type (wrapper type) to char type (Primitive type)

↳ we Cannot Convert Boolean and Character to byte, short, int, long, float, double

↳ xxxValue() method not applicable for Character and Boolean data type.

↳ we Can Convert Boolean and Character to it's Primitive type boolean and char.

→ type in cmd → javap java.lang.Character.

↳ Public char charValue();

③.1 Primitive type to wrapper type

```
Character c1 = new Character('a');
```

③.2 wrapper type to primitive type;

```
char c2 = c1.charValue();
S.o.p(c2) // o/p - a
```

④ Boolean type to boolean type

↳ type in cmd → javap java.lang.Boolean

↳ Public boolean booleanValue();

④.1 Primitive type (boolean) to wrapper type (Boolean)

```
Boolean b1 = new Boolean("true");
```

④.2 Wrapper type (Boolean type) to Primitive type (boolean)

```
boolean b2 = b1.booleanValue();
S.o.p(b2); // true
```

→ Boolean class contains booleanValue() to get boolean Primitive for given boolean object.

③ ParseXXX() : we use ParseXXX() to Convert String type to primitive type.

→ String → ParseXXX() → Primitive type

→ String is object

form-① : here String is input

↳ Public static Primitive ParseXXX(String)

↳ Every wrapper class, except Character class has ParseXXX() to convert String into Primitive type.

① String type to Primitive type (int)

```
int i = Integer.ParseInt("10");
S.o.p(i); // output: 10 (int)
```

② String type to boolean (Primitive type)

↳ Public static boolean ParseBoolean(java.lang.String)

```
boolean b = Boolean.ParseBoolean("Ashish");
S.o.p(b); // false
```

→ Primitive of boolean.

③ When we give String type input instead of integer type it throughs NumberFormatException

```
Integer i1 = Integer.parseInt("ten");
S.o.p(i1);
```

form-② : here input is String and radix

→ Public static Primitive ParseXXX(String s, int radix)

→ range is from 2 to 36

```
int i3 = Integer.ParseInt("100", 2);
S.o.p(i3); // 4
```

⇒ usage of wrapper class in real time coding.

→ wrapper classes are use to perform operations on Command line arguments.

Code-① : Arguments get Concatenate

Command line arguments ⇒ String input

↳ args[0], args[1]

```
S.o.p(args[0] + args[1]);
```

→

⇒ giving arguments from Command line

javac Test.java

java Test Ashish paul

output : Ashishpaul

↳ here 2 arguments get Concatenate

Code-② Program to take 2 inputs from Command line and perform arithmetic operation.

```
int i1 = Integer.ParseInt(args[0]);
int i2 = Integer.ParseInt(args[1]);
S.o.p(i1 + i2); // 30
S.o.p(i1 - i2); // -10
```

→ java Test 10 20 → arguments.

→ When we not give any argument it is zero.

→ argument → we can process in string or we can process converting to Primitive type.

④ To String() :-

→ to convert the wrapper object or Primitive to string.

→ there are 4 form of ~~to~~ tostring() Method.

① form-1

Public String tostring();

① Every wrapper class (including character class) contains the above, tostring() Method to convert wrapper object to string.

② it is overriding version of object class tostring() Method

③ whenever we try to print wrapper object reference initially this tostring() method only Executed.

①.① Wrapper type to string type() :-

```
Integer i = Integer.valueOf("10");
S.o.p(i); // 10
```

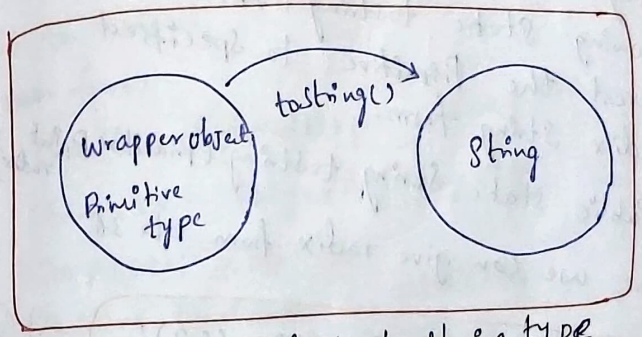
→ Internally it calls tostring() and Print the Data.

② form-2

→ Public static String tostring(Primitive type)

→ Every wrapper class contains a static tostring() Method to convert primitive to string type.

Primitive type → String type



②.① Primitive type(int) to string type

```
String s1 = Integer.toString(10);
System.out.println(s1); // 10
```

②.② Primitive type(boolean) to string type

```
String s2 = Boolean.toString(true);
S.o.p(s2); // true
```

(2.3) Primitive type (character type) to string type

```
String s3 = Character.toString('a');
String s4 = Character.toString('b');
S.o.p(s3); // a
S.o.p(s4); // b
```

(3) form-3

Integer and long classes contains the following static toString() Method to Convert the Primitive to specified radix String form.

⇒ Public static String toString(PrimitiveP, int radix)

→ here we can give radix from 2 to 36.

```
String s5 = Integer.toString(6,2);
S.o.p(s5); // 0110
```

```
String s7 = Integer.toString(7,2);
S.o.p(s7); // 0110
```

↳ here it gives binary form of Primitive type.

(4) form-4 :-

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

(1) Public static String toBinaryString(PrimitiveP);
↳ to Converts primitive type to binary system.

(2) Public static String toOctalString(PrimitiveP);
↳ to Converts Primitive type to octal form

(3) Public static String toHexString(PrimitiveP);
↳ to Convert Primitive type to hexdecimal form.

(1) Primitive type data to binary string

```
String s8 = Integer.toBinaryString(7);
S.o.p(s8); // o/p: 111
```

(2) Primitive type data to octal string.

```
String s9 = Integer.toOctalString(10);
S.o.p(s9); // 12
```

(3) Primitive type to Hex string.

```
String s10 = Integer.toHexString(20);
S.o.p(s10); // 14
```

(4) Primitive type to Hex form

```
String s11 = Integer.toHexString(11);
S.o.p(s11); // b
```

Conclusion :-

① ValueOf() :- To Convert Primitive (or String type to wrapper object.

② xxxValue() :- To Convert wrapper object to Primitive type.

③ ParsexXX() :- To Convert String type to Primitive type.

④ toString() :- Primitive type (or wrapper type to String.

→ we have valueOf() Method in String class to convert Primitive type to String type.

① Public static java.lang.String ValueOf(boolean);
② Public static java.lang.String ValueOf(char);
③ Public static java.lang.String ValueOf(int);
④ Public static java.lang.String ValueOf(long);
⑤ Public static java.lang.String ValueOf(float);
⑥ Public static java.lang.String ValueOf(double);

→

① Static factory Method

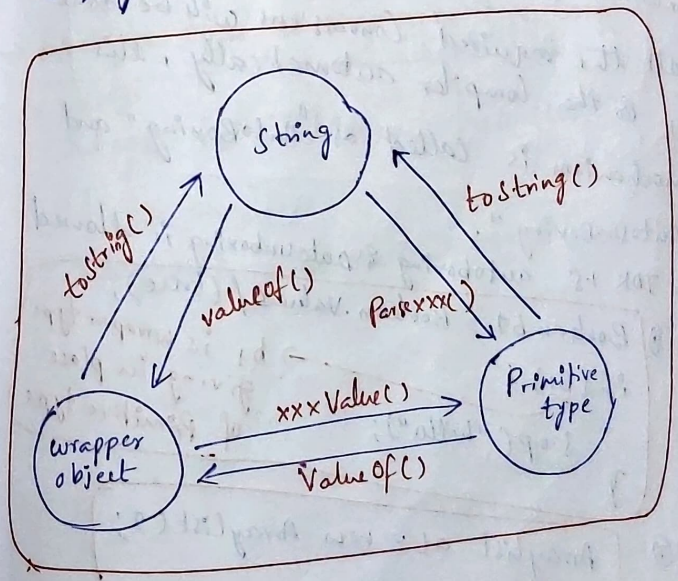
```
String data = String.ValueOf('a');
S.o.p(data); // a
```

→ static because we are using class name.

② Instance factory Method :-

```
String data1 = "Ashis".toUpperCase();
S.o.p(data1);
```

→ instance ~~float~~ because we are using String object.



③ AutoBoxing And AutoUnboxing :

① until JDK 1.4 Version, we can't provide wrapper class objects in place of Primitive and Primitive in place of wrapper object.

→ all the required Conversions should be by the Programmer.

② from JDK 1.5 Version onwards, we can provide primitive in place of wrapper and in place place of wrapper we can keep Primitive.

→ all the required Conversions will be done by the Compiler automatically, this Machanism is called as "AutoBoxing" and "AutoUnBoxing".

→ In JDK 1.5 autoboxing & autounboxing is allowed.

Eg: ①
```
Boolean b1 = Boolean.valueOf(true);
if(b1){
    S.o.p("hello");
}
```
→ b1 is wrapper type giving in place of Primitive type

Eg: ②
```
ArrayList al = new ArrayList();
al.add(10);
```
→ here 10 is primitive type giving in place of wrapper class.

→ Previously in JDK 1.4 it accepts only objects in place of objects.

```
ArrayList al = new ArrayList();
// Convertion of PrimitivetoWrapper done by programmer.
Integer i1 = new Integer(10);
al.add(i1);
S.o.p(al); // [10]
```

→ Primitive type to wrapper type Conversion

↳ 2 ways

```
Integer i2 = Integer.valueOf(10);
        (or)
Integer i3 = new Integer(10);
```

(i) Autoboxing :- Automatic Conversion of Primitive type to wrapper object by the Compiler is Called "Autoboxing".

→ take Primitive type.
→ wrap it around.
→ Keep it up into wrapper object

→ from JDK 1.5 version it is allowed.
→ autoboxing is allowed from JDK 1.5 Version.

Eg:-

```
Integer i4 = 10;   // here 10 is Primitive type
S.o.p(i4);         //10
```

→ behind the seceus Compiler write these code

```
Integer i4 = Integer.Value(10);
```

Note :- Autoboxing is done by the Compiler using a Method Called Value of() Method.

(ii) Auto Unboxing :- Automatic Conversion of wrapper object to Primitive type by Compiler is Called Auto Unboxing.

→ data is wrapped and giving to Primitive type.
→ from JDK 1.5 version autounboxing is allowed

```
Integer i5 = new Integer(10);
int i6 = i5;
```

→ i5 is wrapper type giving to i6 which is primitive type.

wrapper type → Primitive type

→ Compiler Converts Integer to int by using intValue() Method behind the Seceus.
→ behind the seceus Compiler write these Code

```
int i6 = i5.intValue()
```

Note :- Autounboxing is done by the Compiler using a Method Called "xxxValue".

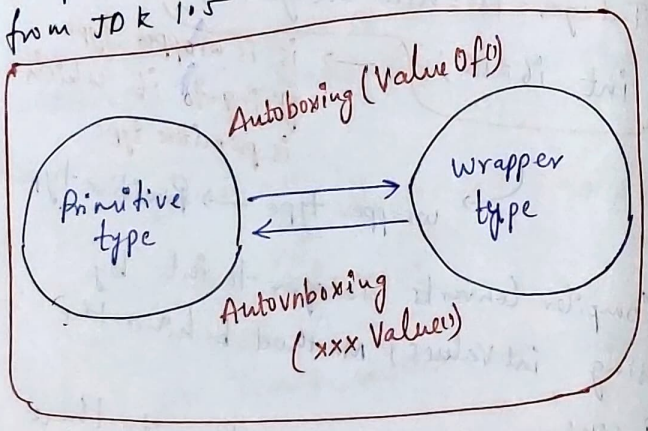
→ Convert 10 to Integer

```
Integer i7 = new Integer(10);
```

(;) it is not inteligent, from JDK 11 it is deprecated.

Integer i8 = Integer.valueOf(10);
↳ static factory Method
↳ it is intelligent.

Deprecate :- don't use this feature
Might remove in future in API.
→ Compiler will do the Conversion automatically from JDK 1.5



Case-①
Compiler is responsible for Conversion of Primitive to wrapper and wrapper to Primitive using the Concept of "Auto Boxing" (or) "Auto Unboxing".

```
Public class Wrapper {
  Static Integer i2=10; // AutoBoxing
  Public static Void main(String[] args)
  {
      int i2=i1;
      m1(i2);
  }
  Public static Void m1(Integer i2) {
      int k=i2; // autoboxing
      S.O.P(k); //10
  }
}
```

Case-②:-

```
Public class Wrapper1{
  Static Integer i1;
Public static Void main(String[] args) {
    int i2=i1;
    S.O.P(i2);
}
}
```

Case-② :- Wrapper-8.java

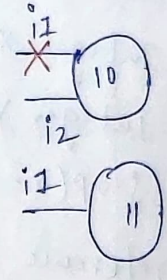
```
Public class Wrapper8{
    Static Integer i1;
    Public Static Void main(String[] args){
        int i2 = i1;
        S.o.p(i2);
    }
}
```

→ at run time JVM assign value to i1 = null
→ JVM will give value for i1.
→ This variable get space in Method area.
→ Method area belongs to heap area.
→ heap area is under control of JVM.

// behind scene Compiler write these code
⇒ int i2 = i1.valueOf()
→ it gives ~~NullPoint~~ NullPointerException

Case-③ :- Wrapper9.java

```
Integer i1 = 10;  // AutoBoxing
Integer i2 = i1;  // AutoBoxing
i1++;             // 11 >> (i1 = i1+1)
S.o.p(i1);        // 11
S.o.p(i2);        // 10
S.o.p(i1==i2);    // false
```



→ wrapper classes are immutable if you make a change it creates new object and Make changes on it.

Case-④ :- 2 new object are created pointed by their respective reference.

```
Integer x = new Integer(10);
Integer y = new Integer(10);
S.o.p(x==y);  // false
```

→ 2 objects pointing by different reference.

Case-⑤ :-

```
Integer x = new Integer(10);
Integer Y = 10;   // AutoBoxing
                  // ↳ behind the secen
                  // Integer Y = Integer.valueOf();
S.o.p(x==Y);
```

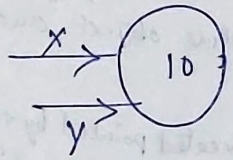
Case-⑥ :-

```
Integer x = new Integer(10);
Integer Y = x;  // Auto Boxing.
S.o.p(x==Y);    // true.
```

Case-6

```
Integer x = new Integer(10);
Integer y = x; // AutoBoxing
S.o.p(x==y); // true.
```

→ reference is reused so pointing to same object.

→ 

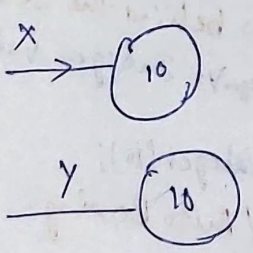
Case-5

```
Integer x = new Integer(10);
Integer y = 10; // AutoBoxing
```

↳ Integer y = Integer.valueOf(10);

```
S.o.p(x==y); // false
```

→ for x is object is created. & for y another object is created.



→ Compiler uses valueOf() for AutoBoxing valueOf() method implemented intelligent way in wrapper classes.

→ for all wrapper classes JVM maintain Buffer of objects.

(4) Buffer of Objects:-

① At the time of loading the .class file JVM will use buffer with objects which are Pre-defined.

② At the time of loading the .class file JVM will create buffer of objects to be used during AutoBoxing of range (-128 to 127).

→ it create buffer of objects with capacity of -128 to 127. this many objects are available for every wrapper classes.

→ Integer x = 10; // AutoBoxing.

(or)

Integer x = Integer.valueOf(10);

↳ factory Methods are intelligent and there is speciality in Memory utilization.

→ Memory utilization of factory Methods.

→ factory Methods Example

```
Integer i = Integer.valueOf(10);
        (or)
Integer i = 10
```

Note:-

① To implement AutoBoxing Concept in wrapper class. a buffer of objects will be Created at the time of class loading.

② During AutoBoxing, if an object has to be created first JVM will check whether the object is already available inside buffer (or) not.

③ if it is available, then JVM will reuse the buffered object instead of Creating new object.

④ if the object is not available inside buffer, then jvm will Create a new object in the heap area, this approach improves the performance and Memory utilization.

⇒ But this buffer of objects Concept is applicable only for few Cases:-

① Byte → -128 to +127

② Short → -128 to +127

③ Integer → -128 to +127

④ Long → -128 to +127

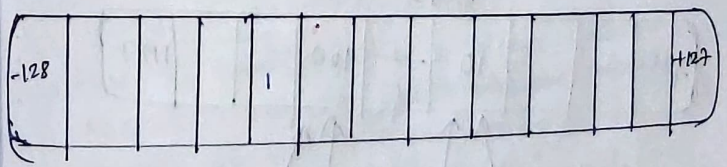
⑤ Character → 0 to 127

⑥ Boolean → true (or) false

⑦ → In the remaining Cases new object will be Created.

→ Compiler uses "valueOf()" for AutoBoxing.
→ valueOf() implemented in intelligent way in wrapper class.

Buffer of objects



-128 to 127

Eg :- ①

```
Integer x = 10;          ——①
Integer y = 10;
System.out.println(x==y); //true.
```

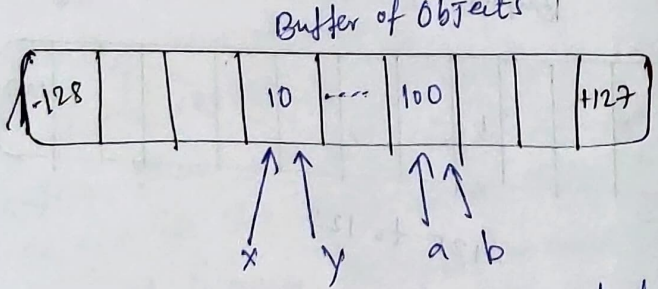
→ x and y Points to in buffer of objects

```
Integer a = 100;         ——②
Integer b = 100;
s.o.p(a==b); //true
```

```
Integer i = 1000;        ——③
Integer j = 1000;
s.o.p(i==j); //false
```

↳ here 1000 not there in buffer of objects. So new object get created.

↳ i & j Points different object Contains 1000.



→ x and y Points to in buffer of objects.

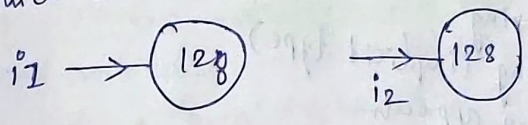
→ a and b Point to 100 in buffer of objects.

Eg ②
```
Integer i1 = 128;
Integer i2 = 128;
s.o.p(i1==i2); //false
```

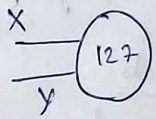
↳ 128 not available in Buffer.

↳ So new objects are Created.

↳ i1 and i2 Points two different object which Contains 128

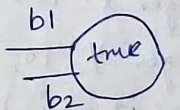


Eg ③
```
Integer x = 127;
Integer y = 127;
s.o.p (a==b); //true.
```



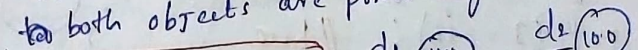
→ 127 is available in Buffer. Same object is reused. both objects pointing to same object.

Eg ④ :-
```
Boolean b1 = true;
Boolean b2 = true;
s.o.p(b1==b2); //true.
```



→ true is available in buffer. Same object is reused. both objects are pointed by b1 & b2

Eg ⑤ :-
```
Double d1 = 10.0;
Double d2 = 10.0;
System.out.println(d1==d2); //false.
```



→ no buffer objects available for double

So new objects are created.