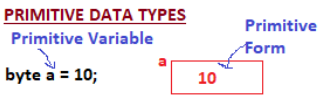
PRIMITIVE DATA TYPES

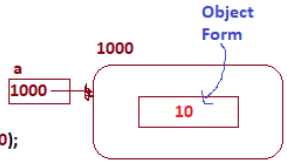



```
byte a = 10;
short b = 150;
int c = 6321;
long d = 21783L;
float e = 45.5f;
double f = 99.75;
char g = 'A';
boolean h= true;
```

Adv: Fast in Execution

Disadv: Makes the application impure Object-Oriented

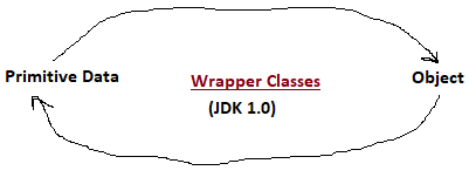
WRAPPER CLASSES



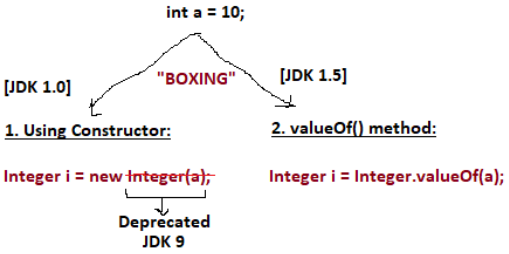

```
Byte a = new Byte((byte)10);
Short b = new Short((short)150);
Integer c = new Integer(6321);
Long d = new Long(21783L);
Float e = new Float(45.5f);
Double f = new Double(99.75);
Character g = new Character('A');
Boolean h = new Boolean(true);
```

Adv: Makes the application 100% Pure Object-Oriented

Disadv: Slow in Execution



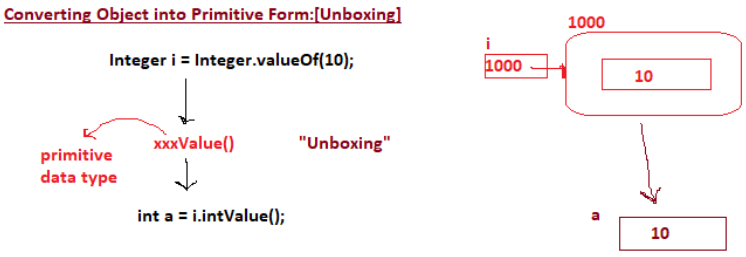
Converting Primitive to Object:[Boxing]




```
class Demo
{
    void disp1()
    {
    }

    static void disp2()
    {
    }
}
```

```
Demo d = new Demo();
d.disp1();

Demo.disp2();
```

```
float a = 45.5f;

Float i = Float.valueOf(a);
```


Converting Object into Primitive Form:[Unboxing]
Integer i = Integer.valueOf(10);
xxxValue()
"Unboxing"
primitive
data type
int a = i.intValue();
1000
i
1000
10
a
10

Auto-Boxing & Auto-Unboxing

int a = 10;

Integer i = Integer.valueOf(a); <OR> Integer i = a;

BOXING

Compiler

Integer i = Integer.valueOf(a);

"AUTO-BOXING"

Integer i = Integer.valueOf(10);

int a = i.intValue(); <OR> int a = i;

"UNBOXING"

int a = i.intValue();

"AUTO-UNBOXING"

Wrapper Classes

BOXING:

```
xxx variable = data;

XXX reference = XXX.valueOf(variable);
```

ClassName

static method

UNBOXING:

```
XXX reference = XXX.valueOf(data);

xxx variable = reference.xxxValue();
```

object reference

non-static method

where,
xxx -> primitive data
XXX -> Wrapper Class

Method Overloading with Boxing and Widening:

1. With Widening Primitive Data Types:

```
class Demo
{
  void disp(long x)
  {
    -----
    -----
  }
}

Demo d = new Demo();
int a = 10;
d.disp(a);
```

2. With Widening Reference Types

```
class Demo
{
  void disp(Number x)
  {
    -----
    -----
  }
}

Demo d = new Demo();
Integer a = 10;
d.disp(a);
```

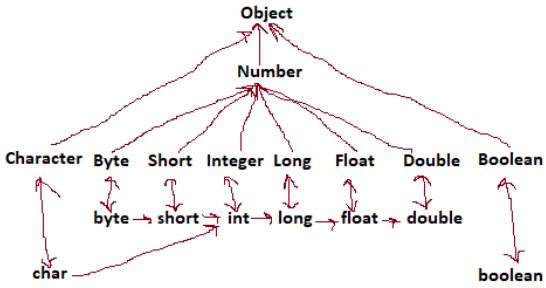
```
class Demo
{
  void disp(Long x)
  {
  }
}

Demo d = new Demo();
Integer a = 10;
d.disp(a);
```

3. With Auto-Boxing

```
class Demo
{
  void disp(Integer x)
  {
  }
}

Demo d = new Demo();
int a = 10;
d.disp(a);
```

4. With Auto-Unboxing:

```
class Demo
{
  void disp(int x)
  {
  }
}

Demo d = new Demo();
Integer a = 10;
d.disp(a);
```



5. With Widening & AutoBoxing

```
class Demo
{
  void disp(long x)
  {
  }

  void disp(Integer x)
  {
  }
}

Demo d = new Demo();
int a = 10;
d.disp(a);
```

Widening > Auto-Boxing

6. With Widening & Auto-Unboxing

```
class Demo
{
  void disp(Number x)
  {
  }

  void disp(int x)
  {
  }
}

Demo d = new Demo();
Integer a = 10;
d.disp(a);
```

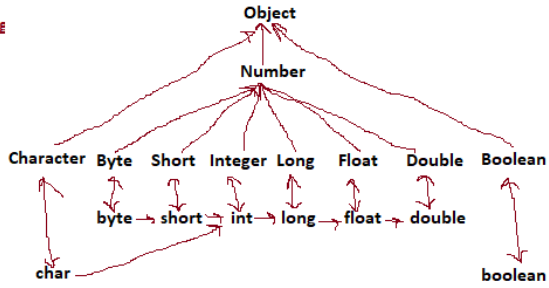
Widening > Auto-Unboxing

7a. Widening & AutoBoxing cannot work tog

```
class Demo
{
  void disp(Long x)
  {
  }
}

Demo d = new Demo();
int a = 10;
d.disp(a);   XCE
```

7b. AutoBoxing followed by Widening c work together

```
class Demo
{
  void disp(Number x)
  {
  }
}

Demo d = new Demo();
int a = 10;
d.disp(a);
```



8. Auto-Unboxing followed by Widening can work together.

```
class Demo
{
  void disp(long x)
  {
  }
}

Demo d = new Demo();
Integer a = 10;
d.disp(a);
```