Working with Java Data Types

Using primitives and Wrapper classes

Primitives

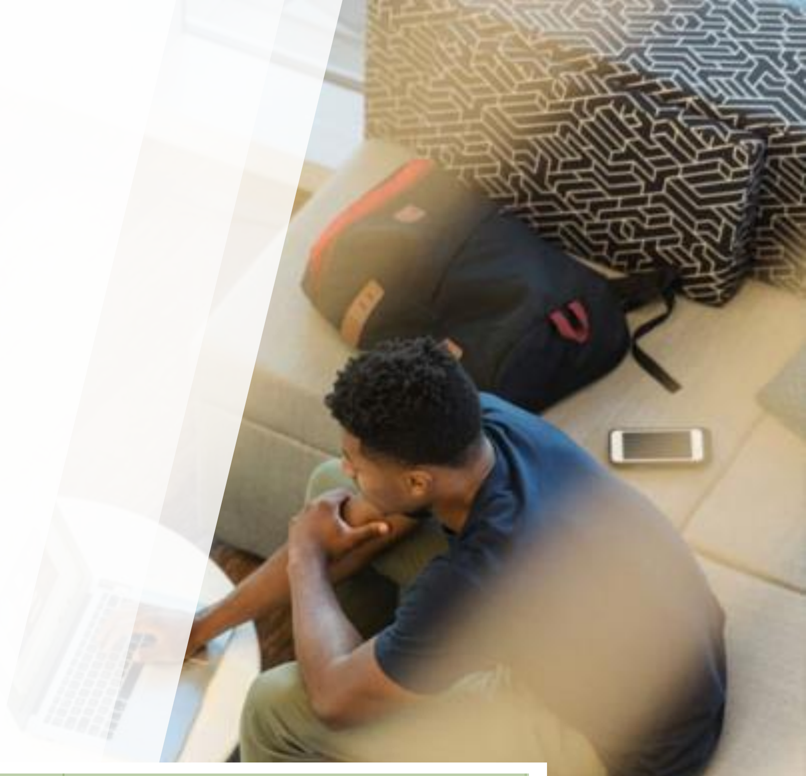
- All numeric types in Java are signed:
  - the leftmost bit (the most significant bit) is used to represent the sign; 1 means negative and 0 means positive.
  - **0**0010011 = **+**19 => 2^7 positive and 2^7 negative values
  - **1**0010011 = **-**19
  - Note: As the decimal number 0 is considered a positive number, it will *appear* as if you have one less positive number in your range. For example, the byte range is -2^7 to 2^7**-1** (-128..+127).
    - the -1 is to allow for 0, which is considered a positive number.
    - -128..-1 is 128 negative numbers; 0..+127 is 128 positive numbers
      - aside: -1..-8 = 8 numbers; 0..7 = 8 numbers also.

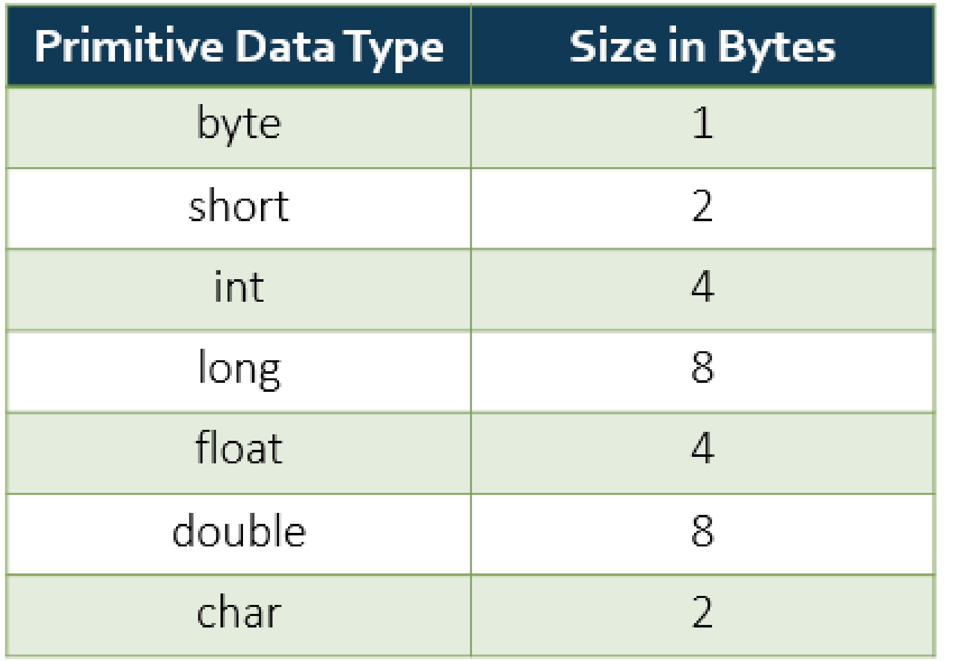
Primitives

- Java has 8 in-built primitive data types:
  - byte (8 bits = 1 byte, -128..+127 inclusive, -2^7 to 2^7-1)
  - short (2 bytes, -32,768..+32,767, -2^{15} to 2^{15}-1)
  - int (4 bytes, -2^{31}..$+2^{31}$-1)
  - long (8 bytes, -2^{63} to 2^{63}-1)
  - float (4 bytes, floating point numbers)
  - double (8 bytes, floating point numbers)
  - boolean (true, false)
  - char (2 bytes <u>unsigned</u>, Unicode, 0..65,535 i.e. 2^{16}-1)
    - can be assigned into *int*, *float*, *double* without a cast
    - assigning to *short* (or vice vearsa) requires a cast

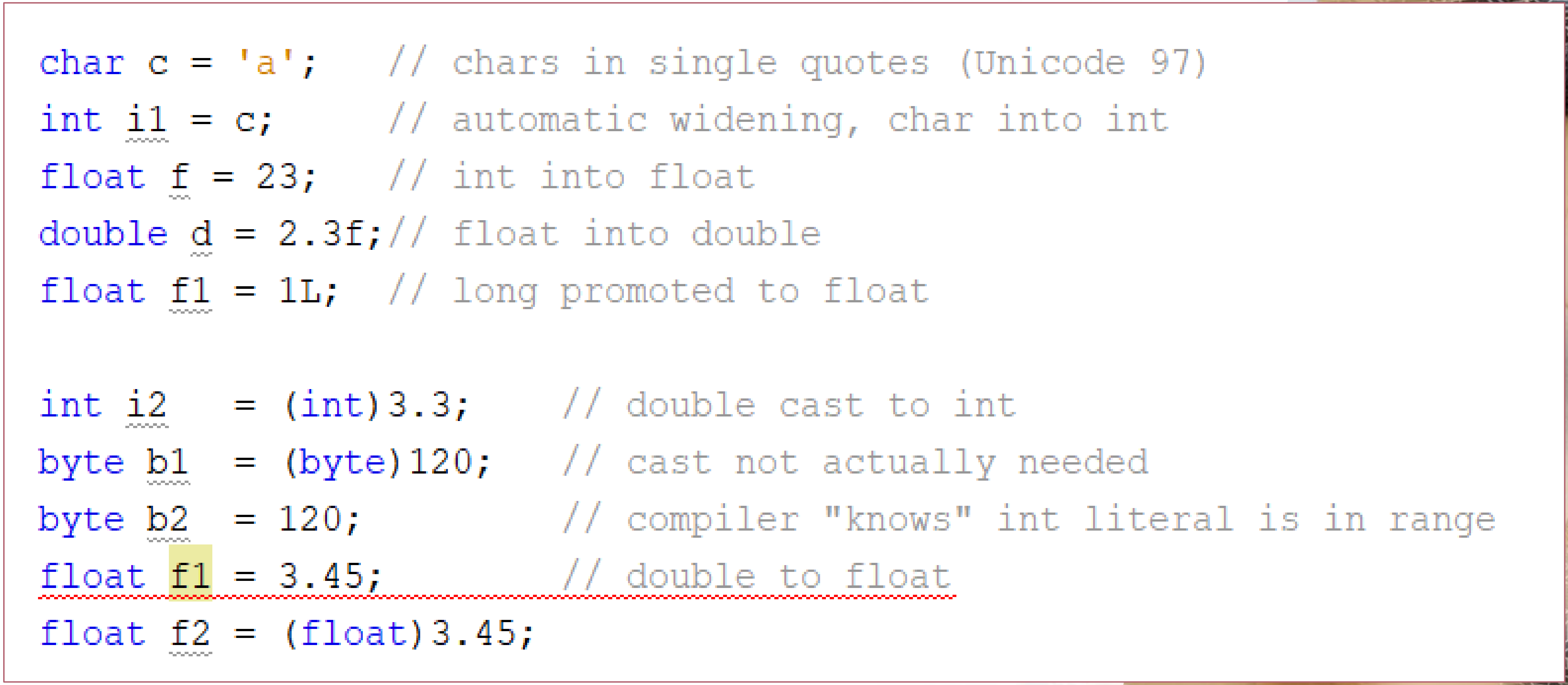
Type Promotion and Casting

- Widening, which is automatic
  - ➢ byte -> short/char -> int -> long -> float -> double
- Narrowing, goes in the opposite direction and requires a cast i.e. where you place the type you wish to cast to, in round brackets
  - ➢double -> float -> long -> int -> short/char -> byte
  - ➢int i = (int)3.3; // double to int



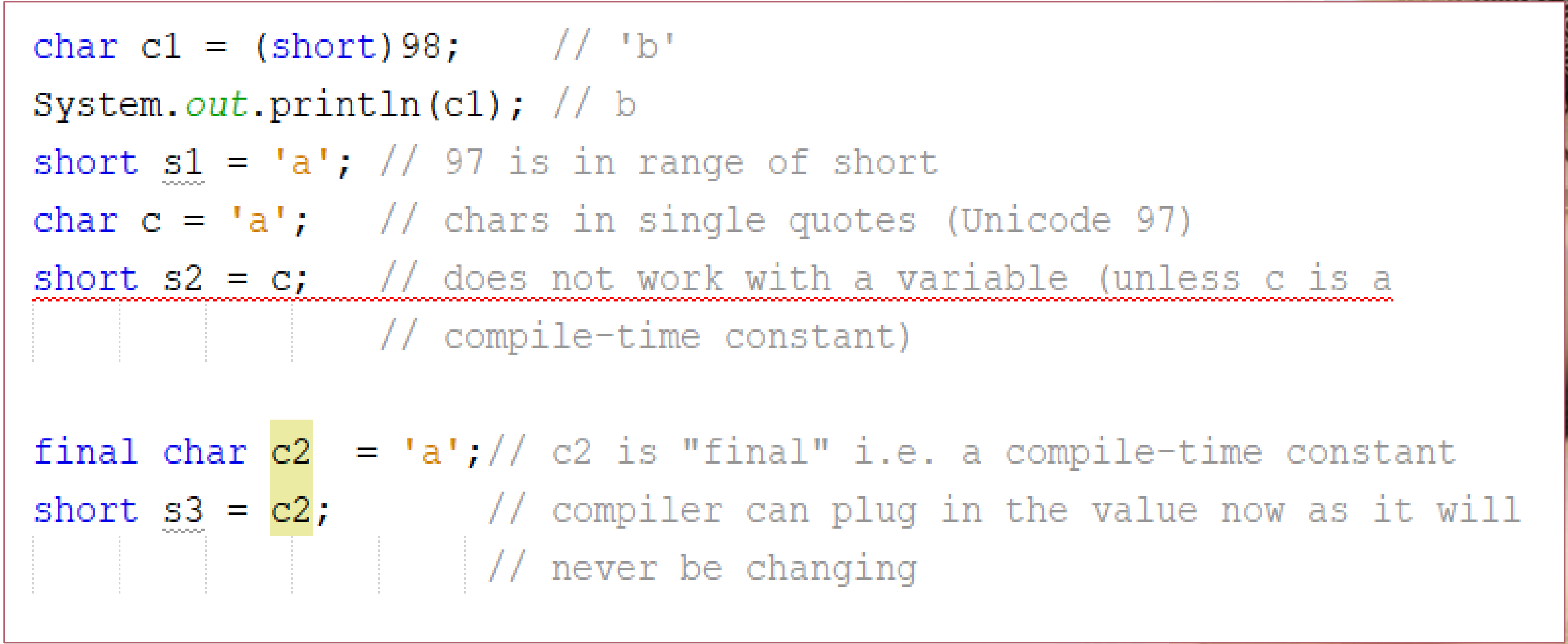


Primitive Data Type	Size in Bytes
byte	1
short	2
int	4
long	8
float	4
double	8
char	2



```
char c = 'a';   // chars in single quotes (Unicode 97)
int i1 = c;     // automatic widening, char into int
float f = 23;   // int into float
double d = 2.3f;// float into double
float f1 = 1L;  // long promoted to float

int i2   = (int)3.3;    // double cast to int
byte b1  = (byte)120;   // cast not actually needed
byte b2  = 120;         // compiler "knows" int literal is in range
float f1 = 3.45;        // double to float
float f2 = (float)3.45;
```



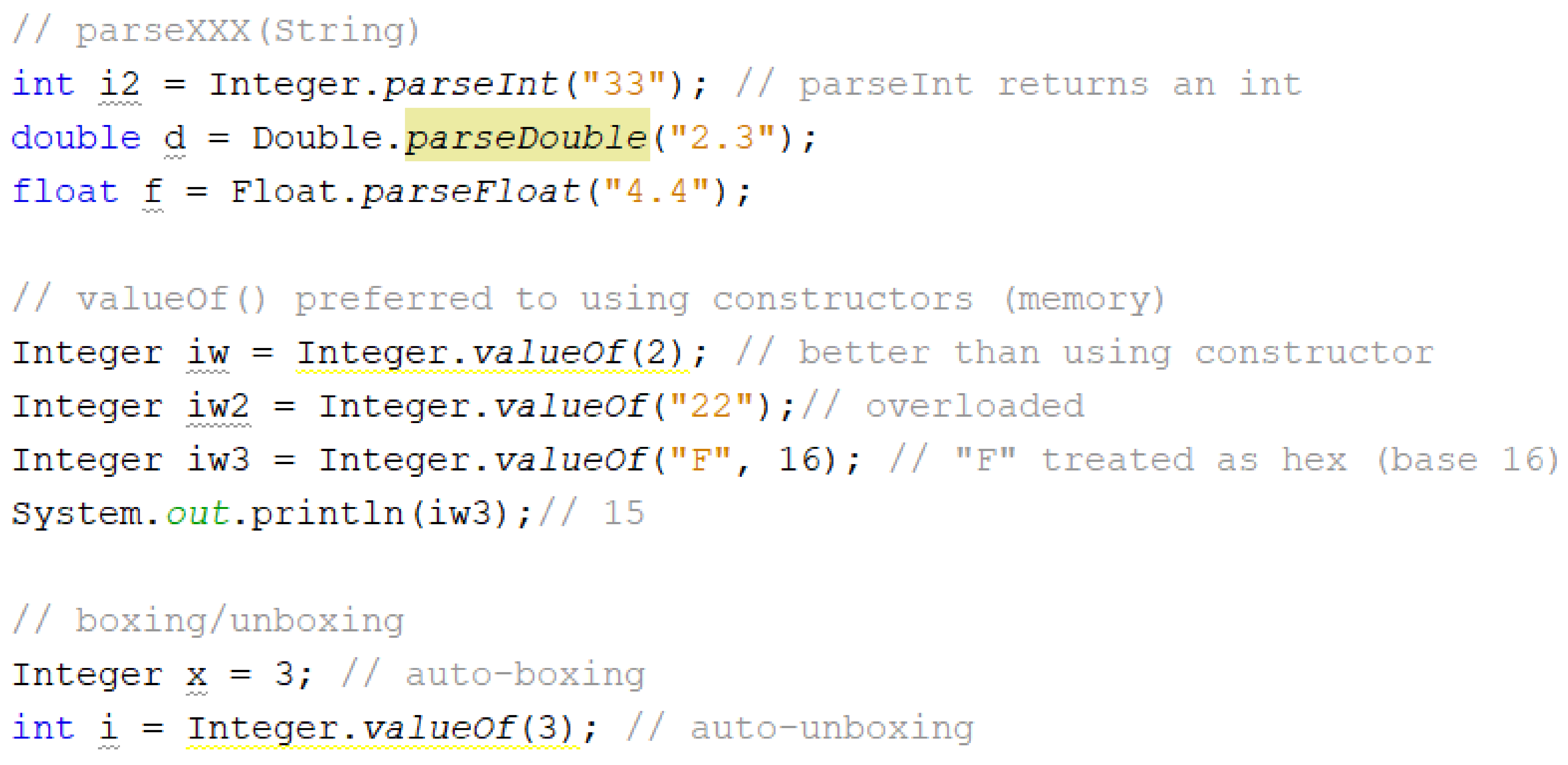
```
char c1 = (short)98;    // 'b'
System.out.println(c1); // b
short s1 = 'a'; // 97 is in range of short
char c = 'a';   // chars in single quotes (Unicode 97)
short s2 = c;   // does not work with a variable (unless c is a
                // compile-time constant)

final char c2  = 'a';// c2 is "final" i.e. a compile-time constant
short s3 = c2;       // compiler can plug in the value now as it will
                     // never be changing
```

Wrappers

- Java provides corresponding Wrapper classes so that we can use primitive data types (int, double etc..) as objects.

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



```
// parseXXX(String)
int i2 = Integer.parseInt("33"); // parseInt returns an int
double d = Double.parseDouble("2.3");
float f = Float.parseFloat("4.4");

// valueOf() preferred to using constructors (memory)
Integer iw = Integer.valueOf(2); // better than using constructor
Integer iw2 = Integer.valueOf("22");// overloaded
Integer iw3 = Integer.valueOf("F", 16); // "F" treated as hex (base 16)
System.out.println(iw3);// 15

// boxing/unboxing
Integer x = 3; // auto-boxing
int i = Integer.valueOf(3); // auto-unboxing
```