Chapter 4

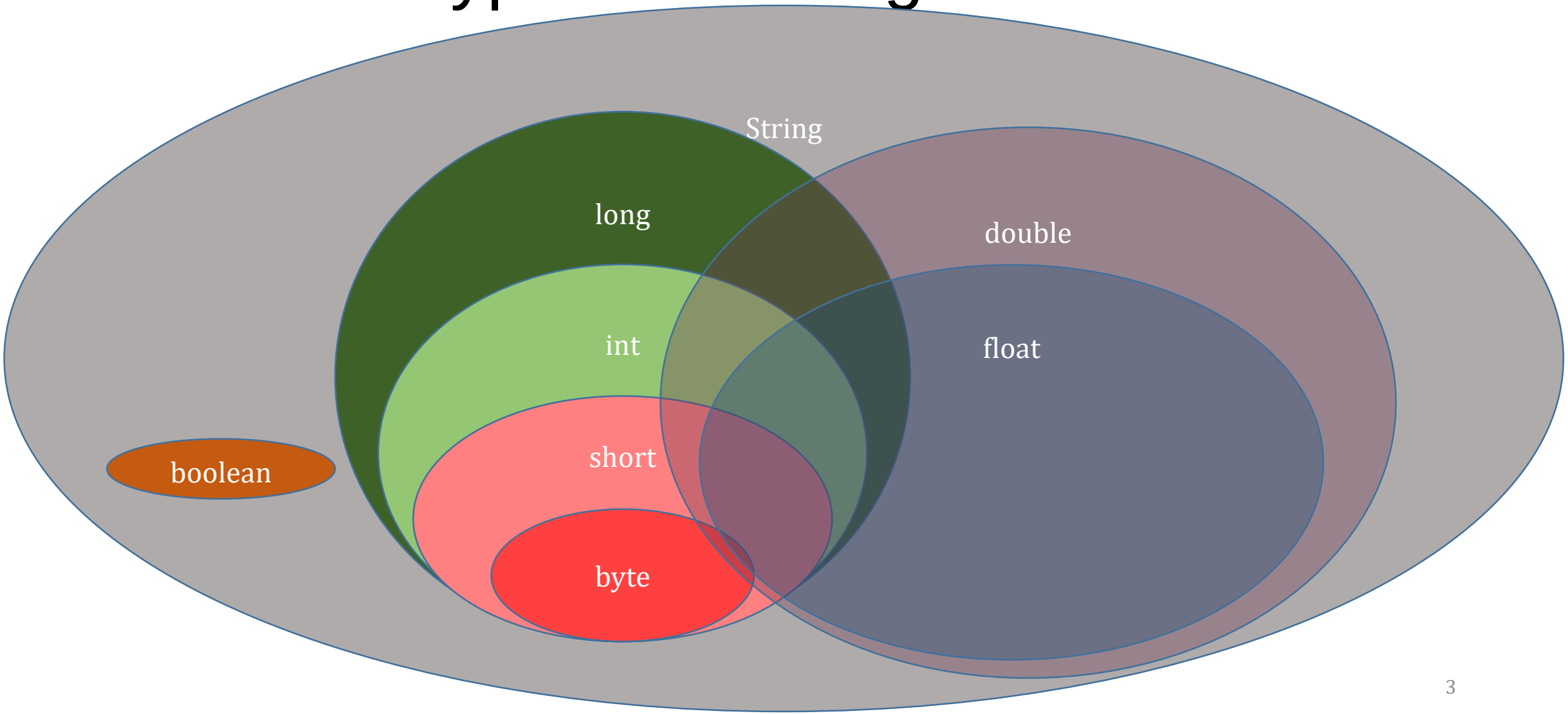
Type Conversion

Under the covers of expressions in Java

Range v. Precision v. Space

Type	Range	Precision	Space
boolean	true/false	Exact	8 bits
byte	+/- 127	Exact	8 bits
short	+/- ~32K	Exact	16 bits
int	+/- ~2M	Exact	32 bits
long	+/- ~ 10^{18}	Exact	64 bits
float	+/- ~ 10^{38}	~15 digits	32 bits
double	+/- ~10^{308}	~23 digits	64 bits

Primitive Type Venn Diagram



Java Type Conversion

```
int i = (int)12.5f; // Casting conversion float to int; compile error without cast
System.out.println("(int)12.5f==" + i); // Convert i to string
float f = i; // int to float, widening conversion
System.out.println("after float widening: " + f); // Convert f to string
System.out.print(f);
f = f * i; // Convert i to float – operation is float*float
System.out.println("*" + i + "==" + f); // Two strings: i and f:
double d = Math.sin(f); // float to double, Math.sin needs a double argument
System.out.println("Math.sin(" + f + ")==" + d); // Two strings: f and d
```

Widening Primitive Conversions

- byte -> short -> int -> long -> float -> double

- Allowed because the result is always correct, no information lost
  - except for loss of precision from integer to floating point

- Integers: Sign-extend on left to get greater width
- Floating point: add ".0" and find closest floating point value

Narrowing Primitive Conversions

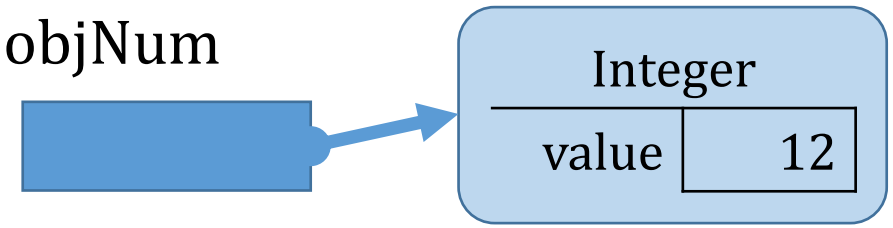
- double -> float -> long -> int -> short -> byte

- Restricted – may lose information
- Compile error without explicit cast
- No runtime error if information lost

- Integer: Truncate bits from left
- Float-> Integer: drop fractional values

The Integer Class

- Static fields: BYTES, MAX_VALUE, MIN_VALUE, SIZE, TYPE
- Single dynamic field: value
- Used when we need a simple object
- static methods for integer utilities
- Dynamic methods for info about value
  - e.g. toString() method to convert 12 to "12".
- Integers are immutable!




```
Integer objNum = new Integer(12); // Always new object…
Integer objNum = Integer.valueOf(12); // May refer to existing
```

Primitive "Boxing"

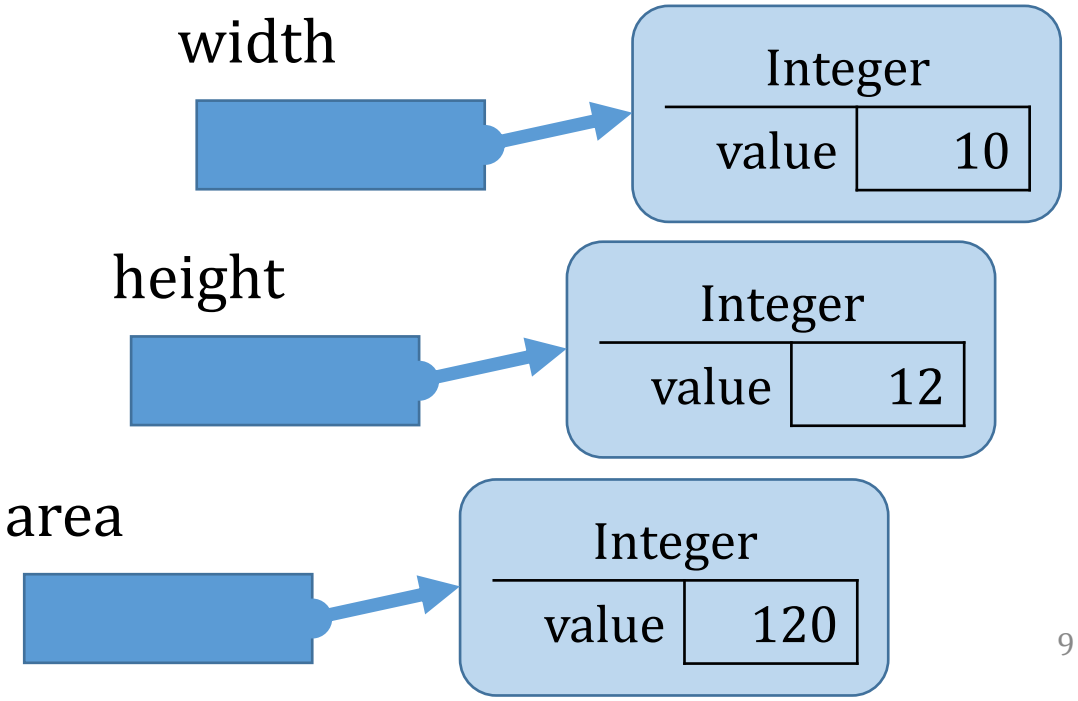
Sect. 7.7.4

- Each primitive type maps to an Object type
  - boolean -> Boolean
  - char -> Character
  - byte, short, int -> Integer
  - long -> Long
  - float -> Float
  - double -> Double
- The object is like a box around the value
- Use? Some Java utilities handle only references, not primitives

Boxing and Unboxing Conversions

- Java infers from the context that boxing or unboxing is required
  - Boxing required when a reference to an object is needed
  - Unboxing required when a primitive value is needed

```
Integer width = 10;
Integer height = 12;
Integer area = width * height;
```



String Conversion

- Values that appear in a String context are converted to String
- Need a reference to do this, so primitives are boxed!
- Objects: Use “toString” method to determine the string representation
- Boolean, Integer, Float, Double toString methods convert values to strings as expected.

- Note: There is a default “toString” method if you don’t code one
  - returns [*package*.]*class*@*address* e.g. conv.Conv@17d99928

Java Conversion Rules

- In a numeric operation, operands are converted to the widest type before evaluation.
- Assignment targets and Parameters are converted to the target type (if that's a narrowing conversion, it's a compiler error!)
- Explicit casts force widening or narrowing conversion.
- When a reference is required, auto-boxing may occur.
- When a primitive is required, auto-unboxing may occur.
- When a String is required, auto-boxing and/or toString is applied