Primitive Types

- Four integer types:
  - byte
  - short
  - int (most common)
  - long

- Two floating-point types:
  - float
  - double (most common)

- One character type:
  - char

- One boolean type:
  - boolean
### Primitive Types, cont.

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Kind of Value</th>
<th>Memory Used</th>
<th>Size Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>integer</td>
<td>1 byte</td>
<td>−128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>integer</td>
<td>2 bytes</td>
<td>−32768 to 32767</td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
<td>4 bytes</td>
<td>−2147483648 to 2147483647</td>
</tr>
<tr>
<td>long</td>
<td>integer</td>
<td>8 bytes</td>
<td>−9223372036854775808 to 9223372036854775807</td>
</tr>
<tr>
<td>float</td>
<td>floating-point</td>
<td>4 bytes</td>
<td>±3.40282347 \times 10^{38} to ±1.40239846 \times 10^{-45}</td>
</tr>
<tr>
<td>double</td>
<td>floating-point</td>
<td>8 bytes</td>
<td>±1.76769313486231570 \times 10^{308} to ±4.94065645841246544 \times 10^{-324}</td>
</tr>
<tr>
<td>char</td>
<td>single character</td>
<td>2 bytes</td>
<td>all Unicode characters</td>
</tr>
<tr>
<td>boolean</td>
<td>true or false</td>
<td>1 bit</td>
<td>not applicable</td>
</tr>
</tbody>
</table>

Display 2.2

Primitive Types
Examples of Primitive Values

- **Integer values:**
  0  -1  365  12000

- **Floating-point values:**
  0.99  -22.8  3.14159  5.0

- **Character values:**
  `a`  `A`  `#`  `

- **Boolean values:**
  true  false
Motivation for Primitive Types

Why are there several different integer types?
- storage space
- operator efficiency

More generally, why are there different types at all?
- reflects how people understand different kinds of data, e.g., letter vs. numeric grades
- makes code more readable (which is a big deal)
- helps prevent programmer errors
Values such as $2$, $3.7$, or 'y' are called constants or literals.

Integer literals can be preceded by a + or - sign, but cannot contain commas.

Every integer literal is either of type `int` or type `long`.

The type of an integer literal can be determined by...looking at it!
An integer literal is of type *long* if it is suffixed with an letter *L* or *l*; otherwise it is of type *int*.

- note that capital *L* is preferred

**Integer literals of type long:**

- 2L
- -372L
- 2147483648l
- 777L
- 1996L
- 0l

**Integer literals of type int:**

- 2
- -372
- 2147483648
- 777
- 1996
- 0
Floating Point Literals

- Floating point literals:
  - Can be preceded by a + or - sign, but cannot contain commas
  - Can be specified in (a type of) scientific notation

- Examples:
  - 865000000.0 can also be written as 8.65e8
  - 0.000483 can also be written as 4.83e-4

- The number in front of the “e” does not need to contain a decimal point, e.g. 4e-4
Floating Point Literal Type

- Every floating point literal is either of type `float` or type `double`.

- The type of a floating point literal can be determined by…looking at it!

- An floating point literal is of type `float` if it is suffixed with an letter `F` or `f`; otherwise it is of type `double`.

- Floating point literals of type `float`:
  - `2.5F`
  - `0.0f`
  - `8.65e8f`
  - `4e-4F`
  - `3f`
  - `+35.4f`
  - `-16F`
  - `-16.0F`
Java is said to be *strongly typed*, which means that there are limitations on mixing variables and values in expressions and assignments.

What is the type of the LHS and RHS for each statement?

```java
int x = 0;
long y = 0;
float z = 0.0f;
int w;

w = x;          // legal; what does it do?
x = y;          // illegal
x = z;          // illegal
y = z;          // illegal
z = 3.6;        // illegal (3.6 is of type double)
y = 25;         // legal, but...why?
```
Sometimes **automatic** conversions between types do take place:

```java
short s;
int x;

s = 83;
x = s;

double doubleVariable;
int intVariable;

intVariable = 7;
doubleVariable = intVariable;
```
In general, a value (or expression) of one numeric type can be assigned to a variable of any type further to the right, as follows:

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

but not to a variable of any type further to the left.

Makes sense intuitively because, for example, any legal byte value is a legal short value.

On the other hand, many legal short values are not legal byte values.
Example – all of the following are legal, and will compile:

```java
byte b = 0;
short s;
int i;
long l;
float f;
double d;

s = b;
i = b;
l = i;
f = l;    // This one is interesting, why?
d = f;
b = 10;
```
Example – NONE (except the first) of the following will compile:

```java
byte b;
short s;
int i;
long l;
float f;
double d;

d = 1.0;       // This one compiles
f = d;
l = f;
i = l;
s = i;
b = s;
```
A type cast creates a value in a new type from an original type.

A type cast can be used to force an assignment when otherwise it would be illegal (thereby over-riding the compiler, in a sense).

Example:

```java
double distance;
distance = 9.0;
int points;
points = distance; // illegal
points = (int)distance; // legal
```
The value of \((\text{int})\text{distance}\) is 9, but the value of \text{distance}, both before and after the cast, is 9.0.

The type of \text{distance} does NOT change and remains \text{double}.

What happens if \text{distance} contains 9.7?

- Any value right of the decimal point is \textit{truncated} (as opposed to \textit{rounded}).
A cast can be performed from any primitive type to any other primitive type, however…

Remember to “cast with care,” because the results can be unpredictable.

```java
int x;
long z = ?; // ? Could be a computation or input
x = (int)z;
```
Arithmetic Operations

- Arithmetic expressions:
  - Formed using the +, −, *, / and % operators
  - Operators have operands, which are literals, variables or sub-expressions.

- Expressions with two or more operators can be viewed as a series of steps, each involving only two operands.
  - The result of one step produces an operand which is used in the next step.
  - Java is left-associative.
  - Most of the basic rules of precedence apply.

- Example:

  ```java
  int x = 0, y = 50, z = 20;
  double balance = 50.25, rate = 0.05;
  
  x = x + y + z;
  balance = balance + balance * rate;
  balance = (balance + balance) * rate;
  ```
An arithmetic expression can have operands of different numeric types.

- \( x + (y \times z) / w \)
- Note that this does not contradict our rules for assignment.

Every arithmetic expression has a (resulting) type.

- \( k = x + (y \times z) / w; \quad // \text{Does this compile?} \)

Given an arithmetic expression:

- If any operand in the expression is of type \texttt{double}, then the expression has type \texttt{double}.
- Otherwise, if any operand in the expression is of type \texttt{float}, then the expression has type \texttt{float}.
- Otherwise, if any operand in the expression is of type \texttt{long}, then the expression has type \texttt{long}.
- Otherwise the expression has type \texttt{int}.
Example:

```java
int hoursWorked   = 40;
double payRate    = 8.25;
double totalPay;
```

Then the expression in the assignment:

```java
totalPay = hoursWorked * payRate
```

is a double with a value of 500.0.
Operators with integer and floating point numbers

- See the program:
  - [http://www.cs.fit.edu/~pbernhar/teaching/cse1001/expressions](http://www.cs.fit.edu/~pbernhar/teaching/cse1001/expressions)