Primitive Types

- **Primitive Types (base types)**
  - Built-in data types; native to most hardware
  - Note: not objects (will use mostly first four)

  - boolean (1 bit)
  - int (4 bytes)
  - double (8 bytes)
  - char (2 bytes)

- **Constants/Literals (by example):**

```java
boolean f = false;
int i = 32769;
double d = 0.333333;
char c = 'x';
```

- byte (1 byte = 8 bits)
- short (2 bytes)
- long (8 bytes)
- float (4 bytes)

```java
byte b = 33;
short s = 21;
long l = 289L;
float = 3.141592F;
```
Named Constants

- **Probably should name most constants**
  - Should have no “Magic Numbers” in program
  - By convention, use all caps in the identifier

- **Use the final keyword**
  - Keeps you from accidentally changing value;
    
    ```java
    final boolean CORRECT = true;
    final int SQUAD_SIZE = 12;
    final double MIN_GPA = 2.5;
    final char FAIL = 'F';
    final String AUTHOR = "Dietolf Ramm";
    ```

- **Note Constants provided by Java: (from API)**
  
  ```java
  Math.PI, Math.E, Integer.MAX_VALUE
  ```
Arithmetic Operators

- **Arithmetic**
  - +, -, *, /, % (remainder or mod)

- **Increment/Decrement**
  - e.g., k++, k--, ++k, --k

- **Logical (results in boolean value)**
  - <, <=, ==, !=, >=, >
  - Used only for numbers except == and !=
  - For boolean only: !, &&, ||

- **String Concatenation**
  - "I’m " + 19 + " years old and live in " + city

- **Assignment**
  - variable = expression
  - variable op= expression
  - (shorthand for: variable = variable op expression)
Operators

- **Arithmetic**
  - +, −, *, /, % (remainder or mod)
  - Work for both integers and reals
  - Except watch / and % for integers
    - What is 13/5? 13%5? 3/5? 3%5?

- **Increment/Decrement**
  - e.g., k++, k--
    - Written as
      - k++; not k = k++; !!
  - Can write code like
    - k = 3 * p++ - m / 5;
  - Usually not a good idea: confuses. Use a separate line for increment of p.
Operators

- Combining Assignment and Arithmetic
  - `variable op= expression`
  - (shorthand for: `variable = variable op expression`)
  - Thus the following lines contain equivalent statements
    
    ```
    k = k - 1;    k -= 1;    k--; 
    q = q * r;    q *= r; 
    s = s / 2;    s /= 2; 
    ```
Assignment

(Familiar by now)

\[ x = 13.3; \]
\[ k = k + 7; \]
\[ area = 2.0 \times \text{Math.PI} \times \text{radius} \times \text{radius}; \]

- Vocalize as “gets” or “becomes”
- Don’t use “equals”: Not Equality

Casting: \text{(type)}

\[ \text{int } k = 3; \text{ double } t; \]
\[ t = k; \quad \text{// OK, No information lost} \]
\[ \text{int } m; \text{ double } s = 3.5; \]
\[ m = s; \quad \text{// ILLEGAL, information lost (accidentally?)} \]
\[ m = (\text{int}) s; \quad \text{// OK, information lost – intent shown} \]
Casting

- **Implicit Casting**
  - Types on two sides of operator differ (**int** and **double**) or (**double** and **int**)
  - Promotes calculation double
  - Can cascade down the expression
  - Compare the following two lines:
    ```
    tempC = 5 / 9 * (tempF + 40) - 40;
    tempF = 9.0 / 5 * (tempC + 40) - 40;
    ```
  - Try for 212°F or 100°C
The Math class

- Contains useful **static** methods
  - **static** means the method does not operate on an object
  - Use class name (Math) rather than object name when using the dot operator.
  - Contains common math functions
  - See Java API
  - Use of some common methods
    ```java
    double x = Math.sqrt(y) * Math.cos(theta);
    long j = Math.round(3.8 * Math.pow(h, n));
    ```