Java Statements

- A Java statement is the smallest unit that is a complete instruction.
- Statements must end with a semi-colon.
- Statements generally contain expressions (expressions have a value)
- One of the simplest is the Assignment Statement

<variable> = <expression>;

For Example:

int height;
height = 34;
The Java Assignment Statement

- The left-hand side must name a single memory location.
- The RHS must have a value;
- The value of the expression on RHS is placed in the memory location named on the LHS.

**CORRECT:** `<variable> = <expression>;`

```java
int width, height;
width = 476;
height = 23;
```

**INCORRECT:** `<expression> = <variable>;`

```java
2 = width;
width + 2 = height;
width + height = 43
```
Expressions

- An expression is something that has a value.
- Literals:
  - integer: 23 67
  - real number: 12.5 -34.455
  - character literal: any single character on keyboard
    - use single quotes 'a' '8' 'H'
  - string literal:
    - use double quotes "oh boy!"
  - boolean literal: true false
- (named) variables:
  - `int width1, width2 = 23;`
  - `width1 = width2;`

Here `width2` can be treated as an expression with value 23.
Initial Values

You must always set the value of a variable before using it. This is referred to as initializing the variable. This can be done at declaration time for primitives or objects:

```java
Player p = new Player();
int width = 23;
```

Or later

```java
int width;
width = 23;
```

What happens if you forget to initialize?

```java
Player p;
p.takeTurn(); // p not initialized
```

Here, p doesn't point to anything so the compiler will give a Null pointer exception.

```java
int area, height, width=4;
area = width * height;
```

Here, height is not initialized. Most compilers (but don't count on it!) will probably initialize height to 0. Some compilers will complain and not even let you compile. Bottom line: you can't predict what will happen - this is bad programming!
Objects

Any object is an expression

The value is the location of the Object. Recall:

Player p1, p2;
p2 = new Player();
p1 = p2;
Here p1 is set to the value of p2.
Arithmetic Expressions

- Integer Arithmetic: +, -, *, e.g. 7 * 5
- Integer Division: /, %

7 / 2 - takes only integer part
7 % 2 - remainder after dividing

- Real Number Arithmetic: +, -, *, /

27.5 + -27.2
Precedence

Which Operator Goes First? see page 94 in text

High Precedence (performed first)

<table>
<thead>
<tr>
<th>()</th>
</tr>
</thead>
<tbody>
<tr>
<td>unary -, +</td>
</tr>
<tr>
<td>* / %</td>
</tr>
<tr>
<td>+ -</td>
</tr>
</tbody>
</table>

Low Precedence (performed last)

Examples:

```java
int number = 5 * 7 - 15 % 6 + 3/4*5 + 3/(4*5) ;
```
Some More Arithmetic Expressions

- Short-Cuts
  - Assignment Operators:
    - `+=, -=, *=, /=, %=`
  - Increment, Decrement, `++`, `--`, Pre and Post

- Relational Expressions
  - Comparison Operators `<`, `>`, `==`, `!=`, `<=`, `>=`
  - Have Boolean Value
  - E.g.
    - `boolean isSmaller = (5 < 8 );`

- Logical Expressions: AND (`&&`), OR (`||`), NOT (`!`)
  - These operate on boolean values
  - AND
    - Truth table:
      - `x` | `y` | `x && y`
      - T  | T  | T
      - T  | F  | F
      - F  | T  | F
      - F  | F  | F
### OR

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x | y</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

### NOT

<table>
<thead>
<tr>
<th>x</th>
<th>!x</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Java is (sort-of) Strongly Typed

This means that if you create variables of different types, you can't always use them together.

Often numbers can be combined but be careful - the results may not always be what you expect!

Example 1:

```java
int age = 45;
char letter = 'k';
letter = age; // WON'T WORK
```

Example 2: Mixing numerical types

```java
int age = 45;
double fraction = 0.4;
fraction = age; // WILL WORK
age = fraction; // WON'T WORK (info can be lost)
```

Can force type change by type casting.

```java
age = (int) fraction; // WILL WORK
```
Constants

- Value must be set at declaration and can't be changed.
- Convention: constants are all upper case

```
final int ID = 1234;
```

- Why use constants?? Suppose you have two unrelated quantities with the same value:
  - `final int SIDES = 6;`
  - `final int RESOLUTION = 6;`

- Why not just use the number 6?
  
  Ans: Makes for code that is easier to modify.

- Why not just have one named constant instead? E.g.

  ```
  final int SIX = 6;
  ```

  Ans: Makes for more understandable code and easier to modify.