Arithmetic Operators
+ Addition
- Subtraction
/ Division (int / floating-point)
    \[ \frac{2}{3} = 0, \frac{2.0}{3.0} = 0.66667 \]
* Multiplication
% Modulus (integer remainder)

Relational/Equality Operators
< Less than
<= Less than or equal to
> Greater than
>= Greater than or equal to
== Equal to
!= Not equal to

Logical Operators
! NOT
&& AND
|| OR

Assignment Operators
= simple assignment
+= addition/assignment
-= subtraction/assignment
*= multiplication/assignment
/= division/assignment
%= modulus/assignment

Increment ++/Decrement -- operators used in prefix and postfix modes
++/-- prefix mode - inc(dec) variable, use variable in the larger expression
++/-- postfix mode - use variable in larger expression, inc(dec) variable

Object Creation: (new) new ObjectCreationName(parameters);

Object Types: value/reference
A value type stores a value of a primitive type. int x = 3;
A reference type stores the address of an object. Circle c = new Circle(2);
A reference variable is created using a class name. myGradeBook = new GradeBook("CIS 182");

Primitive Data Types (Java value types)
Remember: String is a reference type

<table>
<thead>
<tr>
<th>Type</th>
<th>Declaration</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>flag / logical</td>
<td>true, false</td>
</tr>
<tr>
<td>char</td>
<td>character</td>
<td>'A', 'n', '!'</td>
</tr>
<tr>
<td>byte, short, int, long</td>
<td>integral</td>
<td>2, 3, 5000, 0</td>
</tr>
<tr>
<td>float, double</td>
<td>floating-point</td>
<td>123.456, 93</td>
</tr>
</tbody>
</table>

Default numeric literal types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>integral</td>
<td>int x = 3;</td>
</tr>
<tr>
<td></td>
<td>//3 is an int literal</td>
</tr>
<tr>
<td>floating-point</td>
<td>double y = 2.5;</td>
</tr>
<tr>
<td></td>
<td>//2.5 is a double literal</td>
</tr>
</tbody>
</table>

Most commonly used reference type in Java is String. String name = "Jack";

The switch case Construct (break and default are optional)

Form:
```java
switch (expression) {
    case int-constant :
        statement(s);
        [ break; ]
    case int-constant :
        statement(s);
        [ break; ]
    [ default :]
        statement;
}
```

Example:
```java
switch (choice) {
    case 0 :
        System.out.println("You selected 0.");
        break;
    case 1 :
        System.out.println("You selected 1.");
        break;
    default :
        System.out.println("You did not select 0 or 1.");
}
```

The "expression" and "int-constant" are usually type int or char. Java 7 adds the ability to use a string. Use the break keyword to exit the structure (avoid "falling through" other cases). Use the default keyword to provide a default case if none of the case expressions match (similar to a trailing else in an if-else-if statement).

Forms of the if Statement

Simple if
```java
if (expression) {
    statement;
    x++;  
}
```

if/else
```java
if (expression) {
    statement;
    x++;  
} else {
    statement;
    x--; 
}
```

if/else if (nested if)
```java
if (expression) {
    statement;
    x++; 
} else if (expression) {
    statement;
    x--;  
} else if (expression) {
    statement;
    y++;  
}
```

To conditionally execute more than one statement, you must create a compound statement (block) by enclosing the statements in braces (this is true for loops as well):

Form
```java
if (expression) {
    statement;
    x++;  
    System.out.println(x);
}
```

Input using Scanner class

```java
Scanner input = new Scanner(System.in); //keyboard input
input.nextInt(); input.nextDouble();
```

Output methods for System.out or PrintWriter objects
```java
print( ), println( ), printf() [formatted output]
```

Input/Output using JOptionPane class [package javax.swing]
```java
JOptionPane.showMessageDialog(null, "You did not select 0 or 1.");
```

Conversion from a String to a number using Wrapper Classes
```java
double d = Double.parseDouble(dString);
float f = Float.parseFloat(fString);
int  j = Integer.parseInt(sString);
```

Java formatted output [printf()] and String.format() methods
```java
String printf("%10.1f", 12.34567);
```

Java Numeric Conversions and Casts:

Widening conversions are done implicitly:
```java
double x;  
int y = 100; 
//value from y implicitly converted to a double.
```

Narrowing conversions must be done explicitly using a cast:
```java
double x = 100;  
int y = (int) x;  
//value from x explicitly cast to an int
```

In mixed expressions, numeric conversion happens implicitly. double is the “highest” primitive data type, byte is the “lowest”.

Remember to use the methods equals() or compareTo() when comparing Strings rather than relational comparison operators.

String s1 = "abc", s2 = "def";

**String Comparison expressions:**

Compare for equality:
- s1.equals(s2)
- s1.compareTo(s2) == 0

Remember the compareTo() method returns one of 3 values:
- neg number, pos number, 0

Compare for lexical order:
- s1.compareTo(s2) < 0 (s1 before s2)
- s1.compareTo(s2) > 0 (s1 after s2)

Remember to distinguish between integers and real numbers (called floating-point in Java). These are stored differently in memory and have different ranges of values that may be stored.

- integer: 2, 3, -5, 0, 8
- floating-point: 2.0, 0.5, -3.465

Increment ++/Decrement -- operators used in prefix and postfix modes
++/-- prefix mode - inc(dec) variable, use variable in the larger expression
++/-- postfix mode - use variable in larger expression, inc(dec) variable

Object Creation: (new) new ObjectCreationName(parameters);

Input/Output using Scanner class

```java
Scanner input = new Scanner(System.in); //keyboard input
input.nextInt(); input.nextDouble();
```

Output methods for System.out or PrintWriter objects
```java
print( ), println( ), printf() [formatted output]
```

Input/Output using JOptionPane class [package javax.swing]
```java
JOptionPane.showMessageDialog(null, "You did not select 0 or 1.");
```

Conversion from a String to a number using Wrapper Classes
```java
double d = Double.parseDouble(dString);
float f = Float.parseFloat(fString);
int  j = Integer.parseInt(sString);
```

Java formatted output [printf()] and String.format() methods
```java
String printf("%10.1f", 12.34567);
```

Java Numeric Conversions and Casts:

Widening conversions are done implicitly:
```java
double x;  
int y = 100; 
//value from y implicitly converted to a double.
```

Narrowing conversions must be done explicitly using a cast:
```java
double x = 100;  
int y = (int) x;  
//value from x explicitly cast to an int
```

In mixed expressions, numeric conversion happens implicitly. double is the “highest” primitive data type, byte is the “lowest”.

Remember to use the methods equals() or compareTo() when comparing Strings rather than relational comparison operators.

String s1 = "abc", s2 = "def";

**String Comparison expressions:**

Compare for equality:
- s1.equals(s2)
- s1.compareTo(s2) == 0

Remember the compareTo() method returns one of 3 values:
- neg number, pos number, 0

Compare for lexical order:
- s1.compareTo(s2) < 0 (s1 before s2)
- s1.compareTo(s2) > 0 (s1 after s2)

Remember to distinguish between integers and real numbers (called floating-point in Java). These are stored differently in memory and have different ranges of values that may be stored.

- integer: 2, 3, -5, 0, 8
- floating-point: 2.0, 0.5, -3.465
Java Quick Reference Guide  [Jack Wilson – Cerritos College]
Last Update:  Tuesday, February 16, 2016

The **while** Loop (pre-test loop)

Form:  
```
init;  
while (test)  
{  
    statement;  
    update;  
}  
```

Example:  
```
x = 0;  
while (x < 10)  
{  
    sum += x;  
    x++;  
}  
```

The **do-while** Loop (post-test loop)

Form:  
```
do  
{  
    statement;  
    update;  
}  
while (test);  
```

Example:  
```
x = 0;  
do  
{  
    sum += x;  
    x++;  
}  
while (x < 10);  
```

The **for** Loop (pre-test loop)

Form:  
```
for (init; test; update)  
{  
    statement;  
}  
```

Example:  
```
for (int count=1; count<=10; count++)  
{  
    System.out.println( count );  
}  
```

Enhanced **for** loop:

```
for (parameter : collection)  
{  
    statement;  
}  
```

Example:
```
int scores[] = {85, 92, 76, 66, 94};  
for (int number : scores)  
    System.out.println(number);  
```

Selection and Loop Structures

Selection:  
- Unary or single selection
- Binary or dual selection
- Case structure possible when branching on a variable
- Simple selection
  - One condition
- Compound selection
  - Multiple conditions joined with AND / OR operators

Looping:  
- Java Pre-test loops
- Test precedes loop body
  - while
  - for
- Java Post-test loop
- Test follows loop body
  - do-while

Loop Control:  
- 3 types of expressions that are used to control loops:  
  - initialization (init)
  - test
  - update
- Counter-controlled loops, aka definite loops, work with a loop control variable (lcv)
- Sentinel-controlled loops, aka indefinite loops, work with a sentinel value
- Java Loop Early Exit:  
  - **break** statement

Note:  
The **break** statement can be used with a **switch** statement or a **loop** in Java. **Loops** may also use a **continue** statement.

Java Arrays:  
Create an array (2 ways)

1. `<type> <array-name>[ ] = new <type>[size];`
2. `<type> <array-name>[ ] = { <initializer-list> };

//create an array of 20 elements.
int myArray[ ] = new int[20];

//create an array of 3 elements set to the values in the initializer list.
int myArray[ ] = { 1, 2, 3 };  
String stooges[ ] = { "Moe", "Larry", "Curly" };  

//assign value of first element in myArray to the integer variable x.
int x = myArray[0];  

//assign value of the last element in myArray to the integer variable y.
int y = myArray[ myArray.length-1 ];

All arrays have a public field named **length** which holds the number of elements in the array.

Given this declaration:  
```
int x[ ][ ][ ];
```
```
x.length  
```

is the number of elements in the array in the first dimension.
```
x[ ][ ].length  
```

is the number of elements for a specific array in the second dimension.
```
x[ ][ ][ ].length  
```

is the number of elements for a specific array in the third dimension.

Java Methods:  
`<modifier(s)> <type> <method-name> ( [ <type> param1] , [ <type> param2] , … )  

A Java method can return a single value using a **return** statement:  
return <expression>;  
If a method will not return a value, the return type **void** is used in the method header.  
The return statement **return** may be used if needed or left out (causing an **implicit** return at the end of the method).

void printHeadings()  
//no parameters, return type is void
{  
    <method body>  
}

void printDetailLine( String name, int number, double gpa )  
//3 parameters, return type is void
{  
    <method body>  
}

int getCount()  
//no parameters, return type is int
{  
    <method body>  
}

double max( double x, double y )  
//2 parameters, return type is double
{  
    <method body>  
}

When a method is called, the data is passed to the parameters (if any) using **arguments**

//Arguments:  "Jack Wilson", 100, 3.50  
Passed to  
Parameters:  name, number, gpa for Method:
printDetailLine (see method header above):  
printDetailLine( "Jack Wilson", 100, 3.50 );

A method may be declared with one **variable length parameter**. It must be the last parameter declared.
The syntax of the declaration is  
```
<type> ... <parameter-name>.  
```

Examples:  
```
int ... numbers, double ... values, String ... names  
```

//implicit array creation