# I. Increment and Decrement Operators

week 6

- incrementing or decrementing a variable by one is a very common occurrence
- Java provides an operator to "simplify" this for you
- i++; is equivalent to i = i + 1;
- i--; is equivalent to i = i - 1;
- in more detail, Java provides both pre-increment and post increment (and decrement) operators, e.g. ++i, i++, --i, i—
- by itself, ++i and i++ are equivalent**, however** when used with assignment, it gets more complicated

```
j = ++ i;     // pre-increment          j = i++;          //post-increment

i = i + 1;                            j = i;
j = i;                                i = i + 1;
```

- note there are also combination operators, for example

```
balance += amount;      instead of      balance = balance + amount;
items *= 2;             instead of      items = items * 2;
```

- correct usage:

```
i++;
index++;
```

- **incorrect usage – caution:**

  be very careful when using this operator, doing the following will result in unexpected, undesirable behavior

  ```
  i = i++;
  ```

  see java example: incrementExample

# II. ITERATIVE/REPETITIVE CONTROL STRUCTURES

1. indeterminate or indefinite (non-fixed) # steps

   a) **while** loop

```
while (condition)
        statement;              // single statement loop body

while (condition)
{
        statement ;             // multi statement loop body
        statement(s);
        statement affecting condition;
}
```

- this is a pre-test loop
- must pass condition/have a variable set to allow loop entrance
- must change condition value to eventually exit loop

1. test the condition
2. if condition returns a true value
   a) execute loop body
   b) goto step 1

   else exit loop

- summation example

```
int count = 1;
int sum = 0;

while (count <= 5) {
    sum = sum + count;              // sum += count;
    count = count + 1;              // count++;
}
```

b) **do** loop

```
do
    statement;                  // single statement loop body
while (condition);

do
{
        statement ;             // multi statement loop body
        statement(s);
        statement affecting condition;
} while (condition) ;
```

- this is a post-test loop
- don't need any variables set to enter loop body

2. determinate or definite (fixed) # steps

**for** loop BJ 4.3

- given the following common while loop

```
i = start_value;
while (i <= end_value)
{
        statement;
        i++; }
```

- we have a special dedicated loop called a for loop to mimic above

```
for ( i = start_value ; i <= end_value ; i++)
{
        statement(s);
}
```

- steps in for loop operation performed by the loop itself

a) initialize counter to start value
b) check condition, if false, loop terminates
c) execute loop body
d) update counter variable
e) goto step b

**Table 6.2. `for` Loop Examples**

| Loop | Values of `i` | Comment |
|---|---|---|
| `for (i = 0; i <= 5; i++)` | 0 1 2 3 4 5 | Note that the loop is executed 6 times. (See Quality Tip 6.4 on page 235.) |
| `for (i = 5; i >= 0; i--)` | 5 4 3 2 1 0 | Use `i--` for decreasing values. |
| `for (i = 0; i < 9; i = i + 2)` | 0 2 4 6 8 | Use `i = i + 2` for a step size of 2. |
| `for (i = 0; i != 9; i = i + 2)` | 0 2 4 6 8 10 12 14 . . . (infinite loop) | You can use `<` or `<=` instead of `!=` to avoid this problem. |
| `for (i = 1; i <= 20; i = i * 2)` | 1 2 4 8 16 | You can specify any rule for modifying `i`, such as doubling it in every step. |
| `for (i = 0; i < str.length(); i++)` | 0 1 2 . . . until the last valid index of the string `str` | In the loop body, use the expression `str.charAt(i)` to get the `i`th character. |

## III. Simple String Comparison

- String class is not technically a primitive type, but most treat it so (note the capital S, a definition of string will result in an error)
- text in “” is a String type
- String class provides several methods to do String stuff
- strings should **not** be compared using the == operator
- strings should be compared using the string1.equals(string2) method
- example:

  String foo;

  if (foo.equals(“y”))