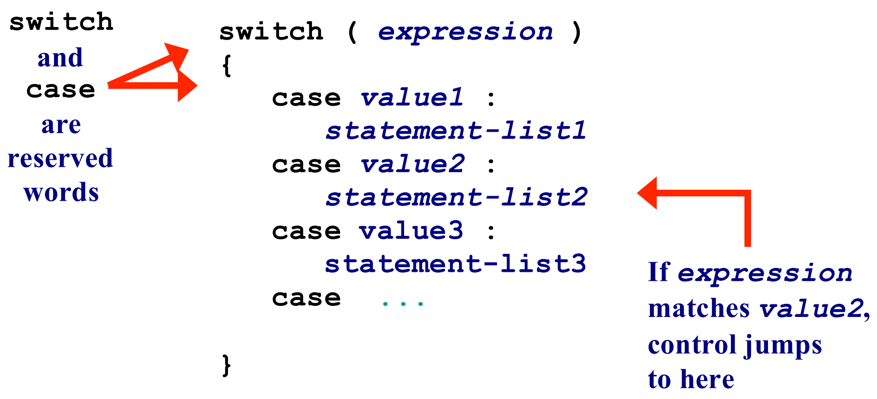
Switch/Case Expressions and Java Shortcuts
Norton CS139

- The **switch/case** Statement
  - A multi-path selection structure
    - The **switch** statement provides another way to decide which statement to execute next
    - The **switch** statement evaluates an expression, then attempts to match the result to one of several possible **cases**
    - Each **case** contains a value and a list of statements
    - The flow of control transfers to statement associated with the first **case** value that matches
  - The general syntax of a switch statement is:



switch and case are reserved words

```
switch ( expression )
{
   case value1 :
      statement-list1
   case value2 :
      statement-list2
   case value3 :
      statement-list3
   case   . . .

}
```

If *expression* matches *value2*, control jumps to here

  - Limiting the flow of control – the **break** statement
    - Often a **break** statement is used as the last statement in each case's statement list
    - A **break** statement causes control to transfer to the end of the **switch** statement
    - If a **break** statement is not used, the flow of control will continue into the next **case**
    - Sometimes this may be appropriate, but often we want to execute only the statements associated with one **case**

- An example of a switch statement:

```
switch ( aCharVar )
{
      case ‘A’:
            aCount++;
            break;
      case ‘B’:
            bCount++;
            break;
      case ‘C’:
            cCount++;
            break;
}
```

- The `default` case
  - A `switch` statement can have an optional `default` *case*
  - The `default` case has no associated value and simply uses the reserved word `default`
  - If the `default` case is present, control will transfer to it if no other case value matches
  - If there is no `default` case, and no other value matches, control falls through to the statement after the `switch`
- The `switch` statement - restrictions
  - Originally, the expression of a `switch` statement had to result in an integral type, meaning an `int` (also `byte` and `short`) or a `char.`
  - However, beginning with Java 1.7 (what we're using), the expression can be a `String` object as well.
  - It cannot, however, be a `boolean` value, a floating point value (`float` or `double`), or a `long` (integral type: why? I have no idea!).
- The `case` statement – limitations
  - The data type of the `case` expression must match that of the `switch` expression.
  - The implicit `boolean` condition in a `switch` statement is equality. (`==` or in the case of a `String`, .`equals()`).
  - You cannot perform relational checks with a `switch` statement
- See `OldGradeReport.java`, `GradeReport.java`, `CharGradeReport.java` and `StringGradeReport.java`.

- Java Shortcuts
    - Incrementers & Decrementers
        - The increment and decrement operators use only one operand
            - The *increment operator* (++) adds one to its operand
            - The *decrement operator* (--) subtracts one from its operand
            - The statement:

              ```
              count++;
              ```

              is (almost) functionally equivalent to:

              ```
              count = count + 1;
              ```

        - Prefix and Postfix forms
            - The increment and decrement operators can be applied in *postfix form*:

              ```
              count++
              ```

            - or *prefix form*:

              ```
              ++count
              ```

            - If used by themselves, the 2 forms are equivalent.
            - When used as part of a larger expression, the two forms can have different effects.
                - Postfix form handles assignment first and increment/decrement second
                - Prefix form handles increment/decrement first and assignment second

              ```
              int a;
              int x = 5;

              a = x++; // a = 5!!!, x = 6
              a = ++x; // a = 6, x = 6
              ```

            - Because of their subtleties, the increment and decrement operators should be used with care

- Assignment Operators
    - Often we perform an operation on a variable, and then store the result back into that variable
    - Java provides assignment operators to simplify that process
    - For example, the statement: `num += count;`
      is equivalent to: `num = num + count;`
    - There are many assignment operators in Java, including the following:

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore: `result /= ( total-MIN ) % num;`
  is equivalent to: `result = result / ( ( total-MIN ) % num );`
- The behavior of some assignment operators depends on the types of the operands
    - If the operands to the += operator are strings, the assignment operator performs string concatenation
    - he behavior of an assignment operator (+=) is always consistent with the behavior of the corresponding operator (+)

- The Conditional Operator
  - Java has a conditional operator that uses a `boolean` condition to determine which of two expressions is evaluated
  - Its syntax is:

```
condition ? expression1 : expression2
```

    - If the condition is true, expression1 is evaluated; if it is false, expression2 is evaluated
    - The value of the entire conditional operator is the value of the selected expression
  - The conditional operator is similar to an if-else statement, except that it is an expression that returns a value
  - For example:

```
     larger = ( ( num1 > num2 ) ? num1 : num2 );
```

    - If `num1` is greater than `num2`, then the value of `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`
  - The conditional operator is ternary because it requires three operands
  - Another example:

```
System.out.println ( "Your change is " + count +
   ( ( count == 1 ) ? "Dime" : "Dimes" ) );
```

    - If `count` equals 1, then "Dime" is printed
    - If `count` is anything other than 1, then "Dimes" is printed