CENG 211 – Programming Fundamentals

Operators, Primitive Data Types & Control Statements

2016 Fall

Arithmetic Operators

Literal	Operation	Example
+	Addition	a + b // Adds a and b
-	Subtraction	a – b // Subtracts b from a
*	Multiplication	a * b // Multiplies a and b
/	Division	a / b // Divides a by b
%	Modulus	a % b // Computes the remainder of a / b

Operator Precedence:

*, /, and % are evaluated first from left to right. Then + and – are evaluated left to right:

a + b * c is equal to a + (b * c), use parenthesis if in doubt or in complex equations.

Logical Operators

Literal	Operation	Example
==	Equal	a == b // True if a equals b
!=	Not Equal	a != b // True if a is not equal to b
<	Less Than	a < b // True if a is less than b
<=	Less Than or Equal	a <= b // True if a == b or a < b
>	Greater Than	a > b // True if a is greater than b
>=	Greater Than or Equal	a >= b // True if a == b or a > b

Warning:
Do not confuse the assignment operator **=** with the equality comparison operator **==**.

Combining Relational Expressions

- || is the disjunction (OR) of two Boolean expressions.
  - true || true → true, true || false → true
  - false || true → true, false || false → false
- && is the conjunction (AND) of two Boolean expressions.
  - true && true → true, true && false → false
  - false && true → false, false && false → false
- Both are short-circuiting,
  - || only evaluates the second argument only if the first one is false.
  - && only evaluates the second argument only if the first one is true.
- ! negates the truth of any Boolean expression
  - !true → false, !false → true

Simple & Compound Assignment

- = is the assignment operator, it returns the value of the expression on the right-hand side.

  System.out.printf("a=5 returns %d\n", a = 5);

  prints ‘a=5 returns 5’

- For binary operators, you can write the following assignment

  <var> = <var> <op> <expr>;

with a compound assignment operator

  <var> <op>= <expr>;

- This works with arithmetic, logical, and bitwise operators.
- Examples:

  a = a + b; // a += b;

  a = a && b; // a &&= b;

Increment and Decrement Operators

- To increment/decrement a variable you can use the ++/-- operator

  a += 1; → ++a; OR a++;

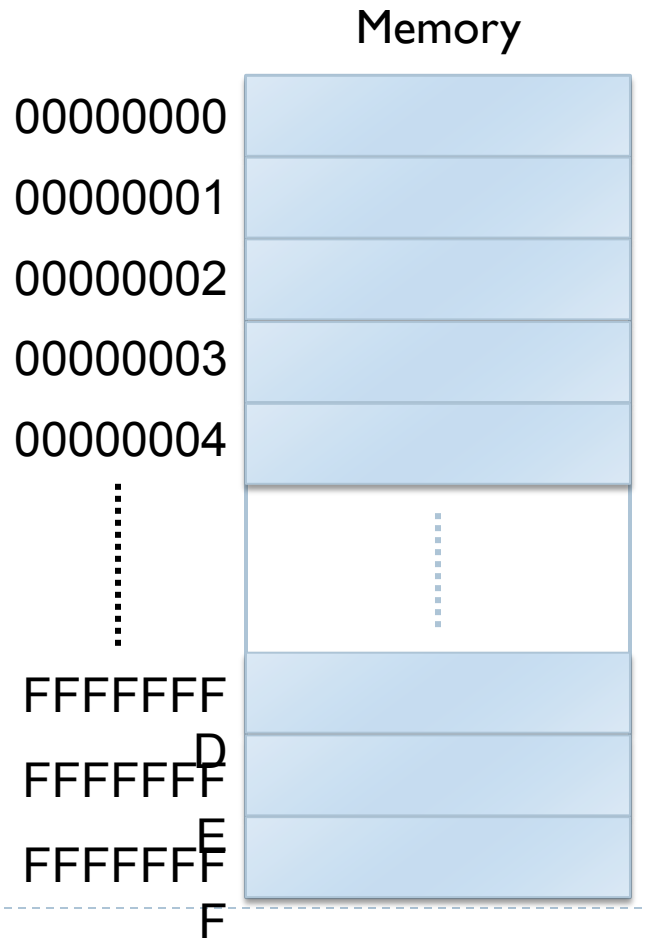
  a -= 1; → --a; OR a--;

- Both the pre-increment (++a) and post-decrement (a++) increment the value of a by one.
  - Pre-increment returns the value of a after incrementing by one.
  - Post-increment returns the original value of a before incremented.
- Example:

```
int a = 5;
int b = ++a; // a equals 6, b equals 6
a = 5;       // reset a to 5
int c = a++; // a equals 6, c equals 5
```

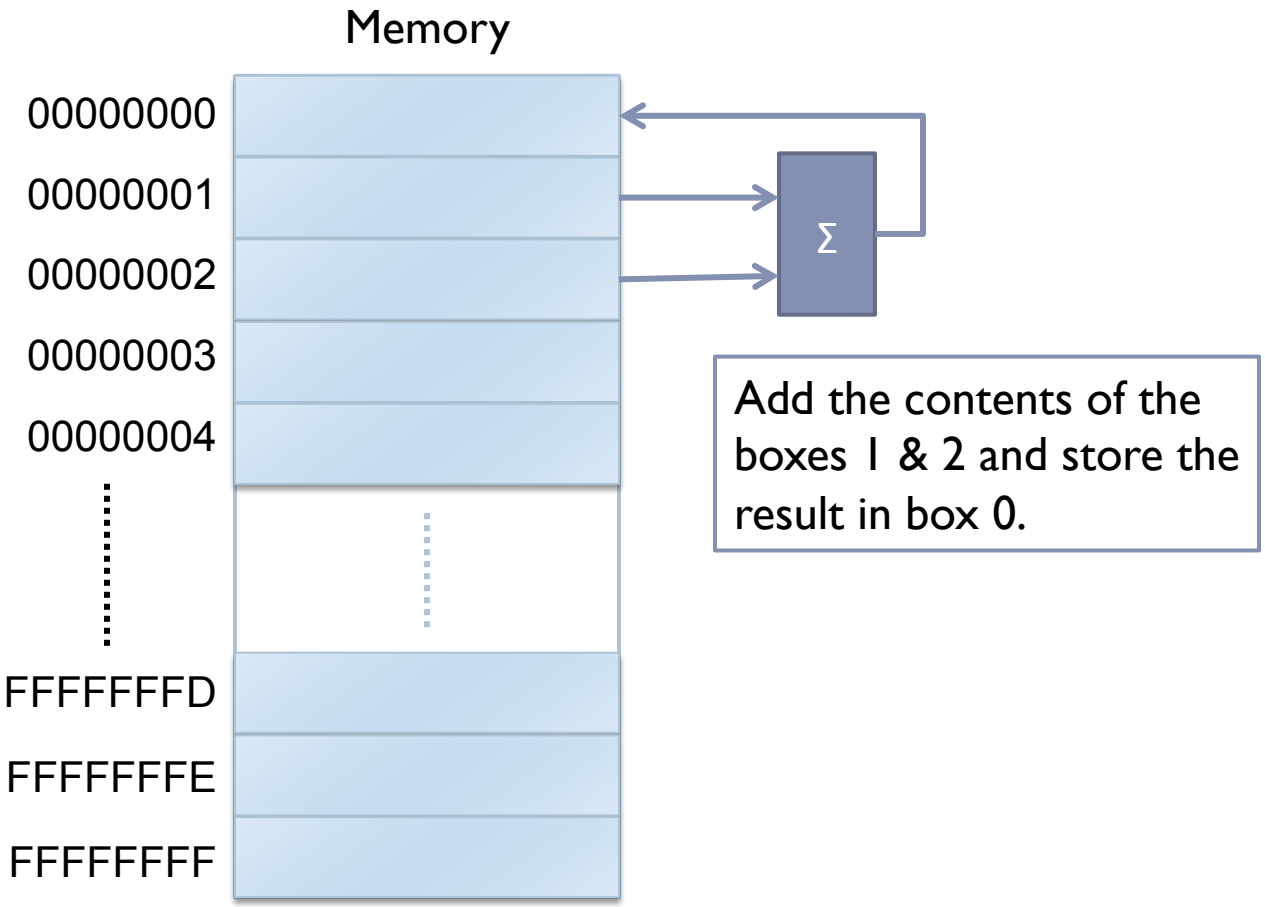
Data Types in Java

- You can imagine your computer memory as a set of numbered boxes:



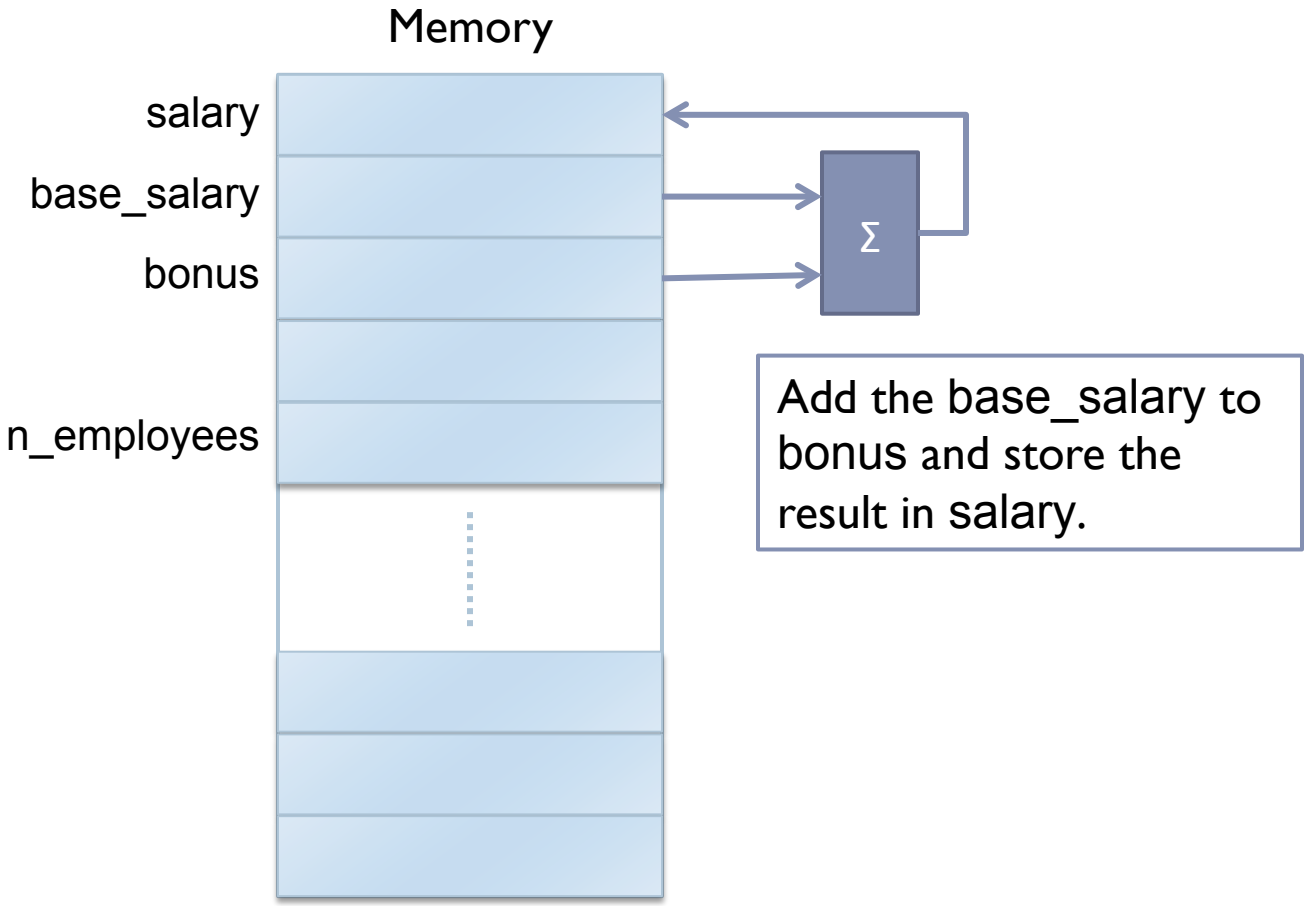
Data Types in Java

- A computer program moves, manipulates, and compares the contents of these boxes, your program data:



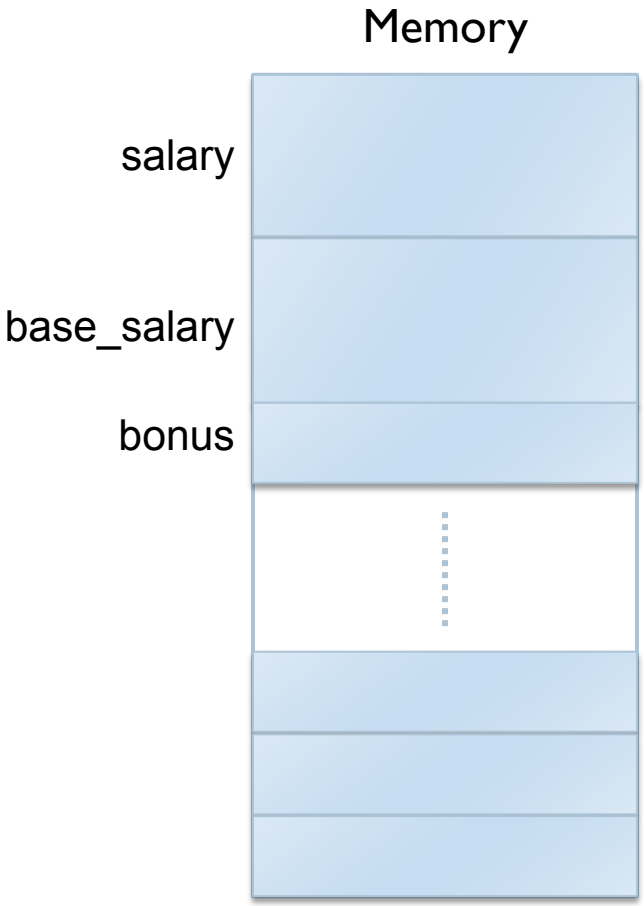
Data Types in Java

- It is difficult to keep track of box numbers, so we use variables to refer to boxes. The compiler assigns each variable to a suitable box.



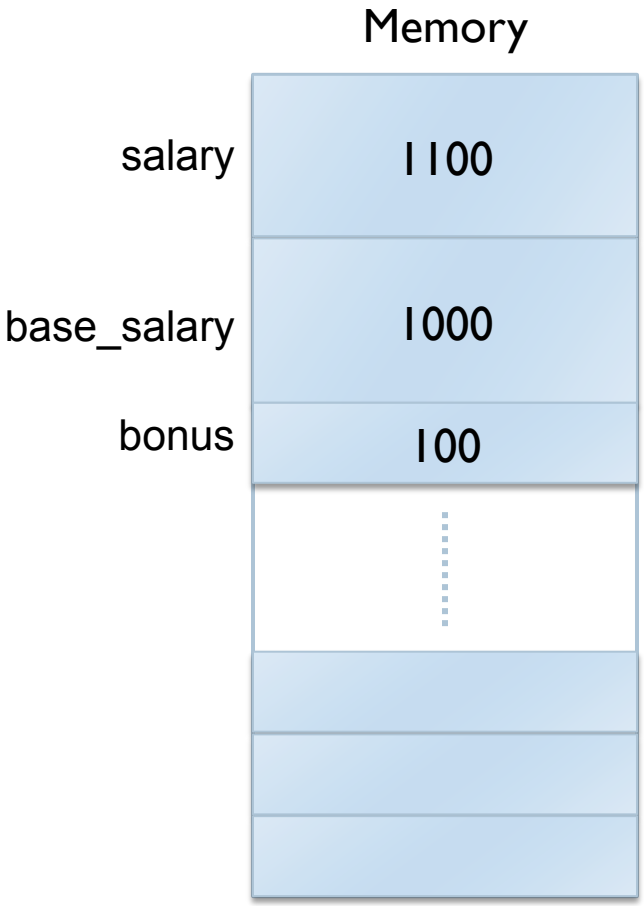
Data Types in Java

- Depending on their data type some variables take up more consecutive boxes than others. This is all handled by the compiler.



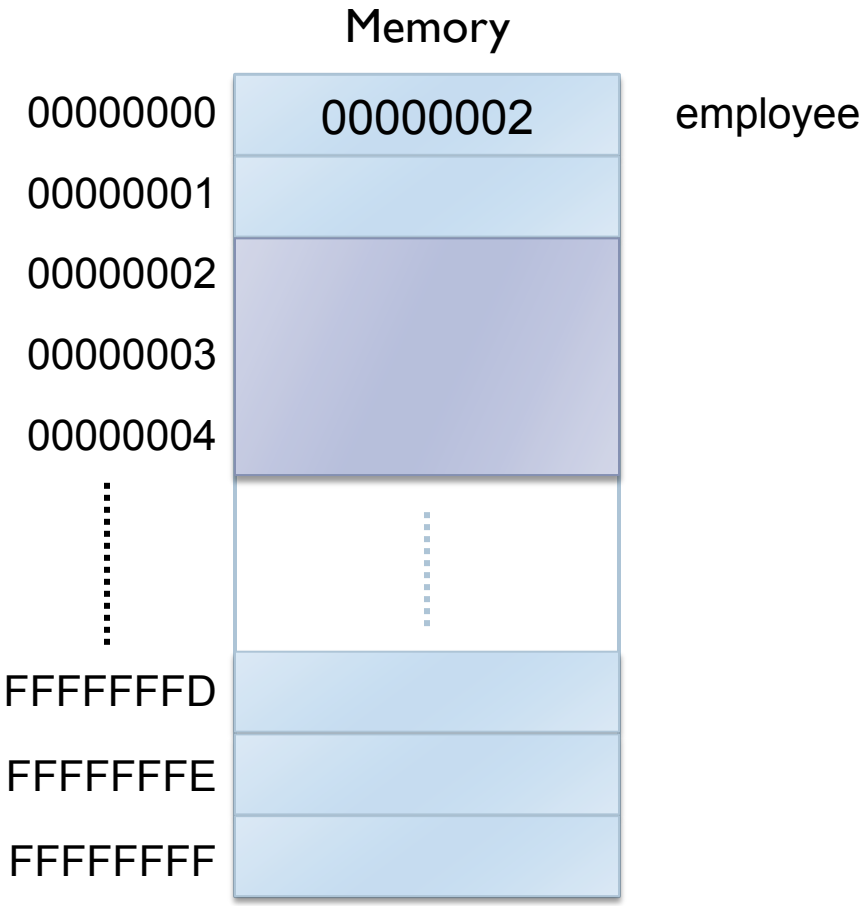
Data Types in Java

- There are two kinds of data types: **Primitive data types** store their contents directly in the boxes that they represent.



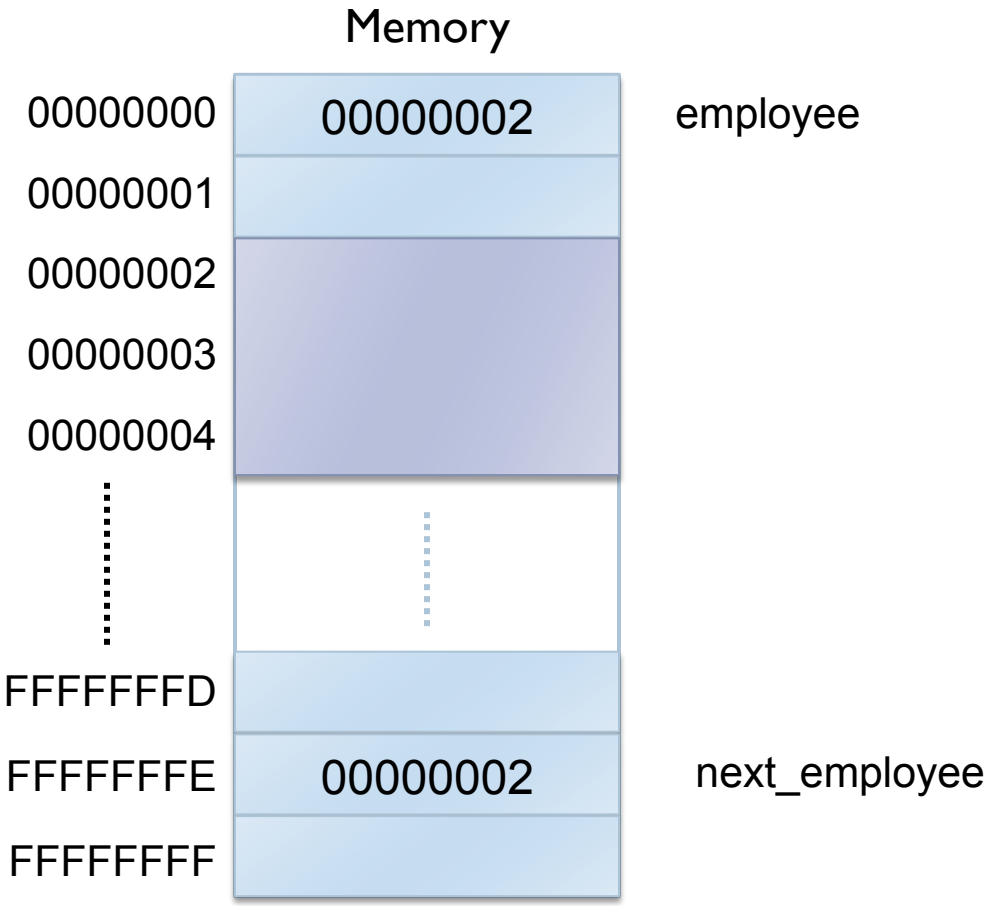
Data Types in Java

- There are two kinds of data types: **Reference data types** store the first box number containing the beginning of actual data.



Data Types in Java

- With **reference data types** we can have multiple variables referring to the same data. We will talk more on references later.



Primitive Data Types

- **boolean**: Truth values, **true** and **false** are keywords that can be used to initialize these.
- **byte**: 8 bit signed integers in the range [-128,127]
- **short**: 16 bit signed integers in the range [-32768, 32767]
- **int**: 32 bit signed integers in the range [-2147483648, 2147483647]
- **long**: 64 bit signed integers in the range [-9223372036854775808, 9223372036854775807]
- **float**: Single-precision 32-bit floating point numbers.
- **double**: Double-precision 64-bit floating point numbers.
- **char**: 16 bit Unicode single characters like 'a' and '0'. Its value is between '\u0000' and '\uffff'.
- There is no **unsigned** keyword in Java.

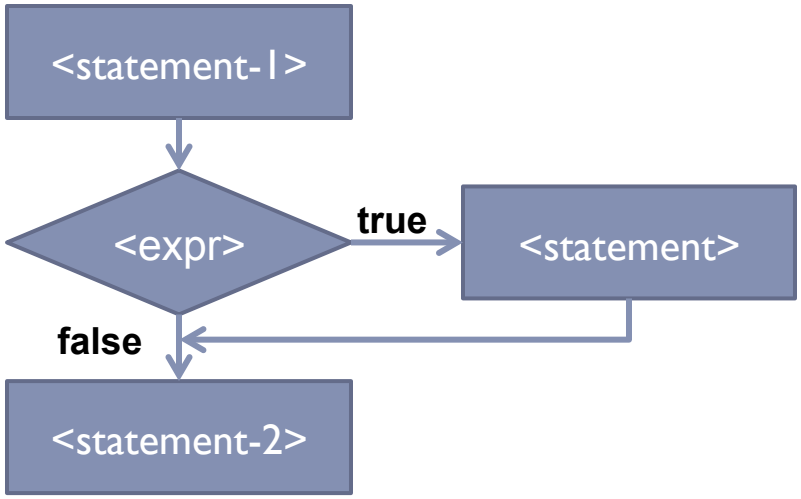
Java Control Statements

- Selection Statements
  - **if** statement
  - **if...else** statement
  - **switch** statement
- Repetition Statements
  - **while** statement
  - **do…while** statement
  - **for** statement
- **break** and **continue** statements

- Note: There is no **goto** statement in Java. However, it is an unused keyword in Java, so you can not use it in your programs.

if Statement

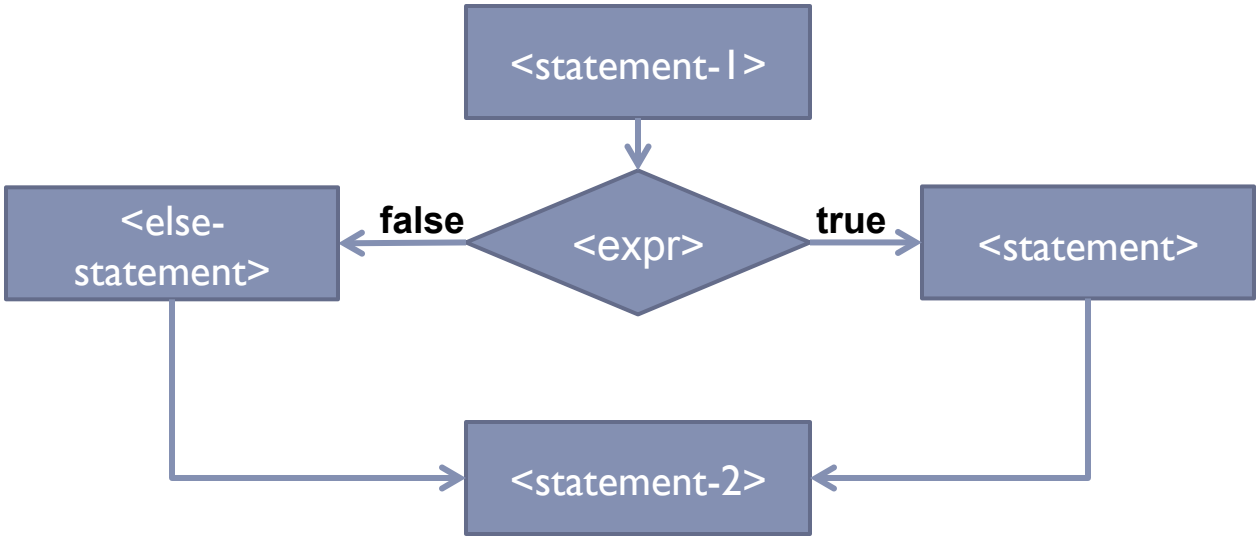
```
<statement-1>
if (<expr>)
    <statement>
<statement-2>
```




```
// Get user input
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the student's age: ");
int age = scanner.nextInt();
// Ensure age is non-negative
if (age < 0)
      age = 0;
computeSchoolYear(age); // Expects non-negative input parameter
```

if..else Statement

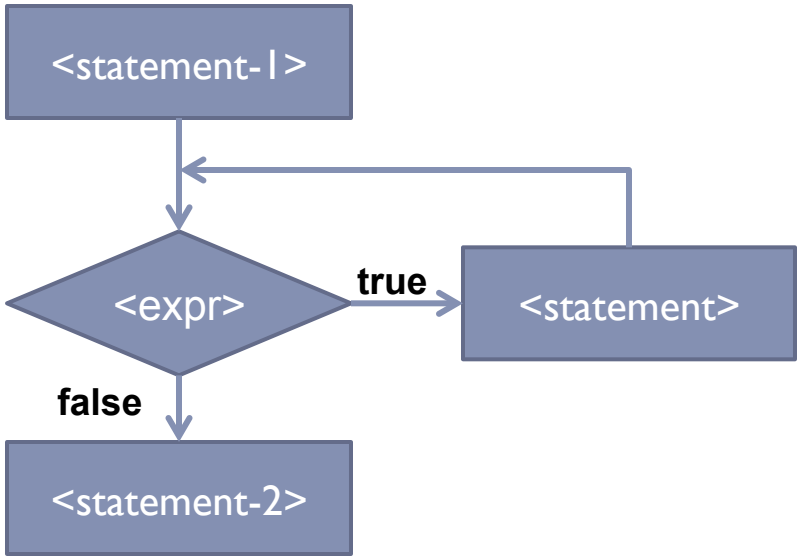
<statement-1>
if (<expr>**)**
    <statement>
else
    <else-statement>
<statement-2>




```
int nApples = basket.size();
if (nApples > 0)
     basket.eatApple(); // Go and eat one.
else // Alert user!
     System.out.println("There no apples left!");
```

while Statement

<statement-1>
while (<expr>**)**
    <statement>
<statement-2>




```
// Get user input
Scanner scanner = new Scanner(System.in);
int age = -1;
while (age < 0) {
      System.out.print("Enter the student's age (age >= 0): ");
      age = scanner.nextInt();
}
computeSchoolYear(age); // Expects non-negative input parameter
```

while Statement Examples

- You can create a loop with a counter:

```
int i = 10;
System.out.printf("Count down...");
while (i >= 0) {
    System.out.printf("%d ", i);
    --i;
}
System.out.printf("\n");
// prints Count down...10 9 8 7 6 5 4 3 2 1 0
```

- You can create a loop with a sentinel:

```
boolean ready = false;
while (!ready) {
    doSomePreparation();
    ready = …;
}
doTheRealThing();
```

while Statement Examples

- You can busy-wait:

```
// At some point checkIfReady() should return true or you
// will loop forever.
callExternalSetupFunc();
while (!checkIfReady())
    ; // Empty statement as while body
doTheRealThing();
```

- You can loop forever:

```
// We will next see how to break out of it conditionally.
while (true) {
    doTheForeverThing();
}
willNeverRun();
```

break Statement

- You can use the break statement to break out of a loop:

```
while (true) {
        System.out.print("Enter a number (-1 to exit): ");
        int number = scanner.nextInt();
        if (number == -1)
                break;
        doSomething(number);
}
```

- In nested loops, it only gets you out of the enclosing loop:

```
int i0 = 5; // Loop counter for the outer loop
while (true) {
        System.out.printf("Outer loop %d\n", i0);
        int i1 = 2; // Loop counter for the inner loop
        while (true) {
                System.out.printf("  Inner loop %d\n", i1);
                if (--i1 < 0) break;
        }
        if (--i0 < 0) break;
}
```

continue Statement

- You can use the continue statement to skip the remaining statements in the loop body for the current iteration:

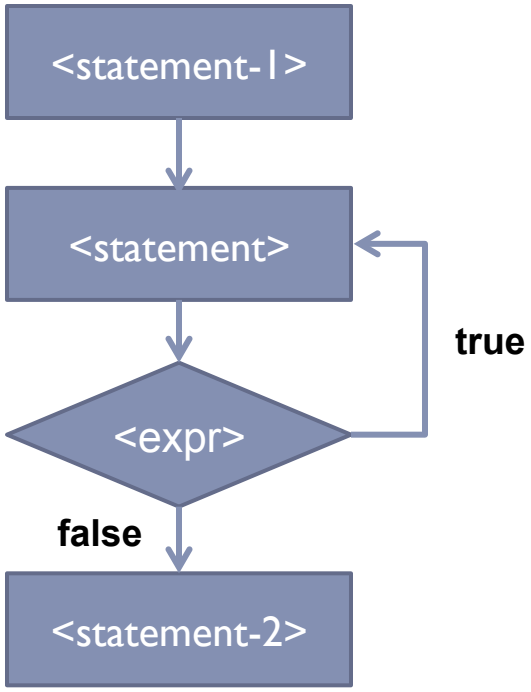
```
int i = 0;
while (i < 10) {
        System.out.println("Always print me.");
        if (i++ > 5)
                continue;
        System.out.println("Print me for i <= 5.");
}
```

- You could eliminate the continue statement above by conditionally running the rest of the loop. If the rest of the loop is small or could be wrapped in a function, try to avoid the continue statement:

```
int i = 0;
while (i < 10) {
        System.out.println("Always print me.");
        if (i++ <= 5) {
                System.out.println("Print me for i <= 5.");
        }
}
```

do…while Statement

<statement-1>
do
    <statement>
while (<expr>**);**
<statement-2>




```
// Get user input
Scanner scanner = new Scanner(System.in);
String folder;
do {
      System.out.print("Enter an existing folder name: ");
      folder = scanner.next();
} while (!folderExists(folder));
saveDocument(folder);
```

A Counter Controlled Loop with while

- To loop over a fixed number of items, you could use a while loop:

```
int counter = 0; // Initialize counter
while (counter < nItems) { // Loop over all item indices
        <Loop Body>
        counter++;
}
```

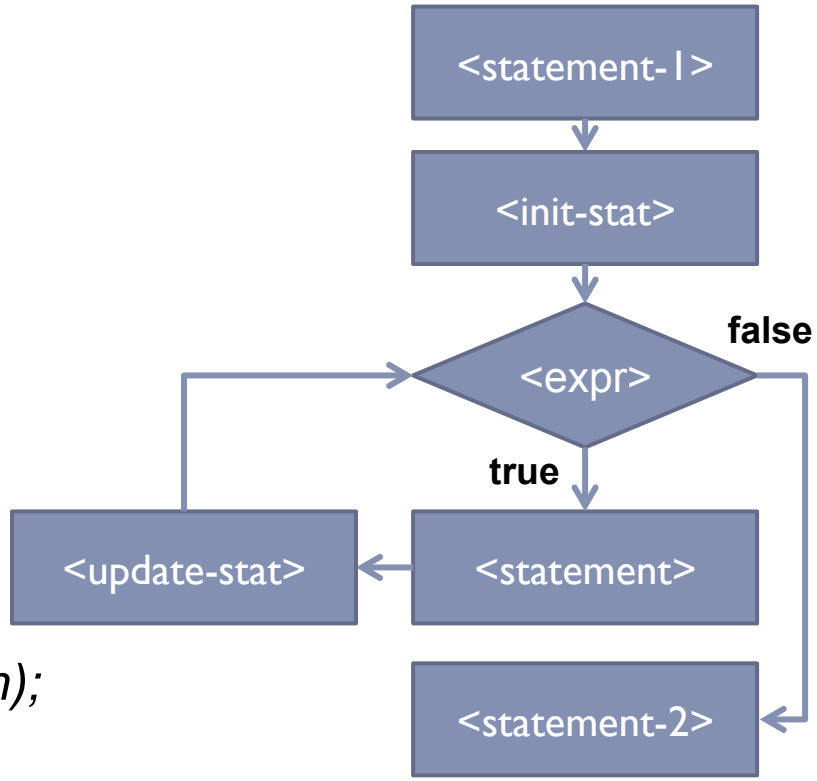
- The loop above goes through indices 0, 1, … , nItems-1.
- What if you want to start at an arbitrary index?
- What if you want to go through only even indices?
- What if you need two counters changing simultaneously?
- It could be generalized as follows:

```
<Initialize Counters>
while (<Continuation Check>) {
        <Loop Body>
        <Update Counters>
}
```

- This is a common pattern and the for statement provides a concise alternative.

for Statement

<statement-1>
for (<init-stat>**;** <expr>**;** <update-stat>**)**
    <statement>
<statement-2>




```
// Get user input
Scanner scanner = new Scanner(System.in);
for (int i = 0; i < 5; i++) {
    System.out.printf("Enter number (%d/5): ", i+1);
    int number = scanner.nextInt();
    doSomething(number);
}
```

“while” to “for” Conversion

- You could convert the while loop from before to a for loop as follows:

```
// while version
int counter = 0; // Initialize counter
while (counter < nItems) { // Loop over all item indices
    <Loop Body>
    counter++;
}
// for version
for (int counter = 0; counter < nItems; counter++) {
    <Loop Body>
}
```

Counting with the for statement

- You can count from 0 to N-1 with either:

```
for (int i = 0; i < N; i++) {
    System.out.printf("%d\n", i);
}
```

- Or:

```
for (int i = 0; i <= N-1; i++) {
    System.out.printf("%d\n", i);
}
```

- Choose one form and stick to it.

Counting with the for statement

- You can count down from N-1 to 0 with either:

```
for (int i = N-1; i >= 0; i--) {
    System.out.printf("%d\n", i);
}
```

- Or:

```
for (int i = N-1; i > -1; i--) {
    System.out.printf("%d\n", i);
}
```

- Choose one form and stick to it.

Counting with the for statement

- You can go over even numbers from 0 to N-1:

```
for (int i = 0; i < N; i += 2) {
    System.out.printf("%d\n", i);
}
```

- You can use two loop counters to go over both even/odd numbers:

```
for (int i = 0, j = 1; i < N; i += 2, j += 2) {
    System.out.printf("%d / %d\n", I, j);
}
```

Counting with the for statement

- You can use the break statement in for loops:

```
for (int i = 0; i < N; i++) {
        System.out.printf(Enter up to %d integers (-1 to stop): ", N);
        int number = scanner.nextInt();
        if (number == -1)
              break;
        doSomething(number);
}
```

```
> java LoopTest
Enter up to 5 integers (-1 to stop): 1
Doing sth with 1
Enter up to 5 integers (-1 to stop): 1
Doing sth with 1
Enter up to 5 integers (-1 to stop): -1
>
```

Counting with the for statement

- You can use the continue statement in for loops:

```
for (int i = 0; i < N; i++) {
        System.out.printf(Enter %d integers (-1 to skip): ", N);
        int number = scanner.nextInt();
        if (number == -1)
              continue;
        doSomething(number);
}
```

```
> java LoopTest
Enter 5 integers (-1 to skip): 1
Doing sth with 1
Enter 5 integers (-1 to skip): 1
Doing sth with 1
Enter 5 integers (-1 to skip): -1
Enter 5 integers (-1 to skip): 1
Doing sth with 1
Enter 5 integers (-1 to skip): -1
>
```

Nested for Loops

- You can nest for loops:

```
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 2; j++) {
        System.out.printf("i, j = %d, %d\n", i, j);
    }
}
```

```
> java LoopTest
i, j = 0, 0
i, j = 0, 1
i, j = 1, 0
i, j = 1, 1
i, j = 2, 0
i, j = 2, 1
i, j = 3, 0
i, j = 3, 1
```

Nested for Loops

- You can nest for loops:

```
for (int i = 0; i < 5; i++) {
    for (int j = 0; j <= i; j++) {
        System.out.print(“*");
    }
    System.out.println();
}
```

```
> java LoopTest
*
**
***
****
*****
```

Nested if..else Statements

- Sometimes you need to take action depending on a variable with a set of possible values:

```
System.out.print("Enter an integer in the range [1, 4]: ");
int number = scanner.nextInt();
if (number == 1)
    doTheFirstThing();
else if (number == 2)
    doTheSecondThing();
else if (number == 3)
    doTheThirdThing();
else if (number == 4)
    doTheFourthThing();
else
    System.out.printf("Number out of range!!!\n");
```

Nested if..else Statements

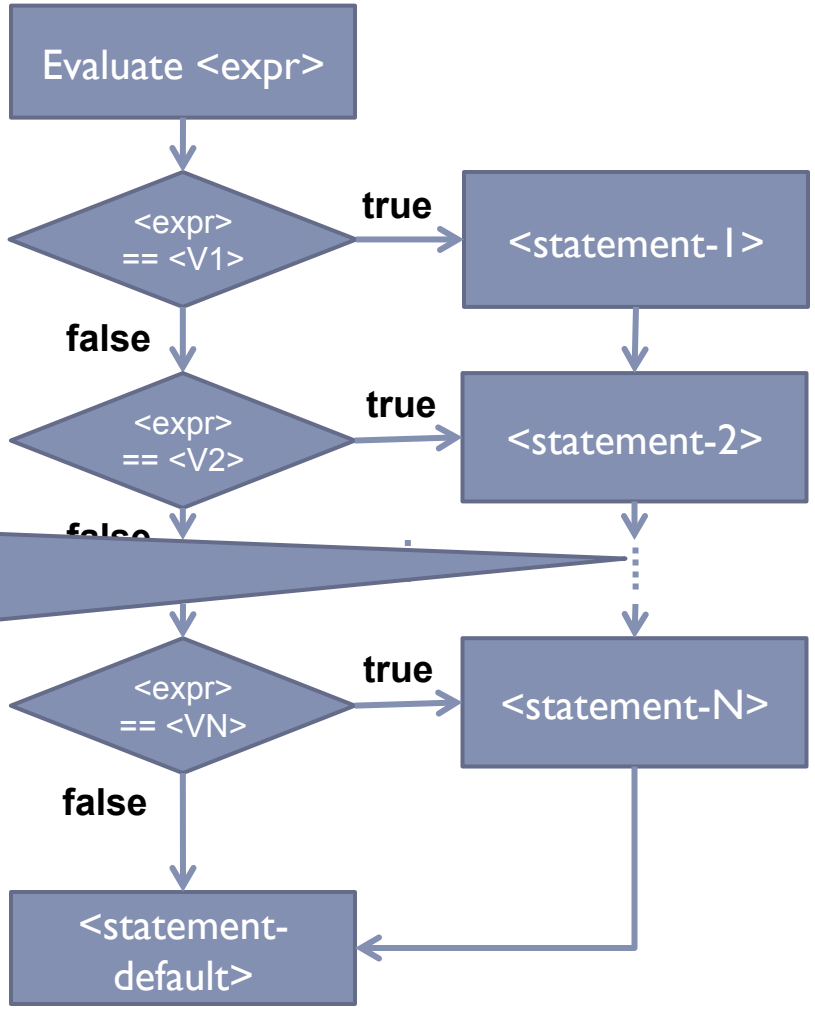
- Sometimes you need to take action depending on a variable with a set of possible values:

```
System.out.print("Enter an integer in the range [1, 4]: ");
int number = scanner.nextInt();
if (number == 1)
    doTheFirstThing();
else if (number == 2)
    doTheSecondThing();
else if (number == 3)
    doTheThirdThing();
else if (number == 4)
    doTheFourthThing();
else
    System.out.printf("%d is not in the range [1, 4]!!!\n", number);
```

switch Statement

```
switch (<expr>) {
case <V1>: <statement-1>
case <V2>: <statement-2>
...
case <VN>: <statement-N>
default:   <statement-default>
}
```

Fall-through behavior of switch causes all the remaining statements to execute once a match is found.



switch Statement with Fall-Through

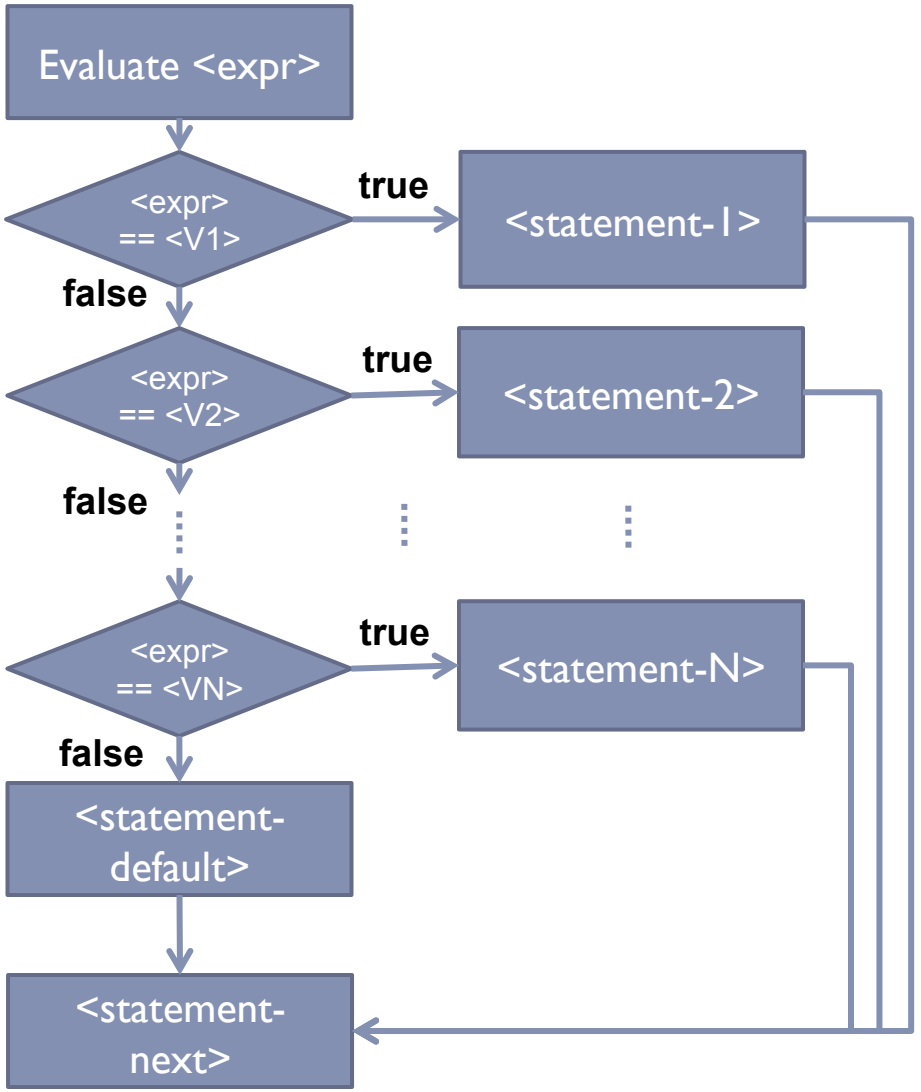
- We can not directly convert the nested if's to a switch because of fall-through behaviour:

```
System.out.print("Enter an integer in the range [1, 4]: ");
int number = scanner.nextInt();
switch (number) {
case 1: doTheFirstThing();
case 2: doTheSecondThing();
case 3: doTheThirdThing();
case 4: doTheFourthThing();
default:
        System.out.printf("%d is not in the range [1, 4]!!!\n", number);
}
```

```
> java SwitchTest
Enter an integer in the range [1, 4]: 1
Doing the first thing
Doing the second thing
Doing the third thing
Doing the fourth thing
1 is not in the range [1, 4]!!!
>
```

switch Statement with breaks

```
switch (<expr>) {
case <V1>:
        <statement-1>
        break;
case <V2>:
        <statement-2>
        break;
…
case <VN>:
        <statement-N>
        break;
default:
        <statement-default>
        break;
}
<statement-next>
```



switch Statement with breaks

- We can now convert the nested if's to a switch:

```
System.out.print("Enter an integer in the range [1, 4]: ");
int number = scanner.nextInt();
switch (number) {
case 1: doTheFirstThing();  break;
case 2: doTheSecondThing(); break;
case 3: doTheThirdThing();  break;
case 4: doTheFourthThing(); break;
default:
        System.out.printf("%d is not in the range [1, 4]!!!\n", number);
        break;
}
```

```
> java SwitchTest
Enter an integer in the range [1, 4]: 1
Doing the first thing
> java SwitchTest
Enter an integer in the range [1, 4]: 3
Doing the third thing
> java SwitchTest
Enter an integer in the range [1, 4]: 6
6 is not in the range [1, 4]!!!
>
```

switch Statement – Merging cases

- We can use the fall-through to merge cases:

```
System.out.print("Enter an integer in" +
                                "the range [1, 4]: ");
int number = scanner.nextInt();
switch (number) {
case 1:
case 2:
        doTheFirstThing();
        doTheSecondThing();
        break;
case 3:
case 4:
        doTheThirdThing();
        doTheFourthThing();
        break;
default:
        System.out.printf("%d is not in" +
                " the range [1, 4]!!!\n", number);
        break;
}
```

```
> java SwitchTest
Enter an integer in the range [1, 4]: 2
Doing the first thing
Doing the second thing
> java SwitchTest
Enter an integer in the range [1, 4]: 3
Doing the third thing
Doing the fourth thing
> java SwitchTest
Enter an integer in the range [1, 4]: -1
-1 is not in the range [1, 4]!!!
>
```

switch Statement – String as expression

- In *Java 7 SE* and later you can use a String in the switch statement:

```
System.out.print("Enter \"One\" or \"Two\": ");
String str = scanner.next();
switch (str) {
case "one":
case "One":
case "ONE": doTheFirstThing(); break;
case "two":
case "Two":
case "TWO": doTheSecondThing(); break;
default: System.out.printf("%s is not \"One\" or \"Two\"!!!\n", str); break;
}
```

We will later see how to convert strings to lowercase/uppercase. It is much better to convert to a common form and then write the cases for that form.

```
> java SwitchTest
Enter "One" or "Two": one
Doing the first thing
> java SwitchTest
Enter "One" or "Two": TWO
Doing the second thing
>
```

switch Statement – Expression Constraints

- The switch expression HAS TO evaluate to an integer (or a String in Java 7 and later). For other expressions you have to use nested if statements:

```
System.out.print("Enter a number: ");
String str = scanner.next();
double number = Double.parseDouble(str);
switch (number) {// THIS IS AN ERROR!
case 3.14159: System.out.println("You have entered PI"); break;
case 2.71828: System.out.println("You have entered E"); break;
default:
    System.out.printf("I do not know %f!!!\n", number);
    break;
}
```

```
> javac SwitchTest.java
SwitchTest.java:10: error: possible loss of precision
      switch (number) {
             ^
  required: int
  found:    double
1 error
>
```

switch Statement – Why integers?

- The switch statement with an integer expression can be implemented with a jump table or binary search.
- This can be more efficient than using a nested set of if..else..if comparisons.
- In Java 7, strings in the switch/case statements are converted to their hash-codes, so that they can be treated as integers.

switch Statement – A Dense Jump Table

```
switch (i) {
case 0: S0(); break;
case 1: S1(); break;
case 2: S2(); break;
case 3:
case 5: S3_5(); break;
}
```

Pseudocode:

```
JumpTable[] = { ?, ?, ?, ?, ?, ? };
1: if (i < 0 || i > 3) goto 11;
2: goto JumpTable[i];
3: S0();
4: goto 11;
5: S1();
6: goto 11;
7: S2();
8: goto 11;
9: S3_5();
10: goto 11;
11:
```

switch Statement – A Dense Jump Table

```
switch (i) {
case 0: S0(); break;
case 1: S1(); break;
case 2: S2(); break;
case 3:
case 5: S3_5(); break;
}
```

Pseudocode:

```
JumpTable[] = { 3, 5, 7, 9, 11, 9 };
1: if (i < 0 || i > 3) goto 11;
2: goto JumpTable[i];
3: S0();
4: goto 11;
5: S1();
6: goto 11;
7: S2();
8: goto 11;
9: S3_5();
10: goto 11;
11:
```

switch Statement – A Sparse Table

```
switch (i) {
case 0: S0(); break;
case 10: S10(); break;
case 24: S24(); break;
case 303: S303(); break;
case 402: S402(); break;
case 512: S512(); break;
case 668: S668(); break;
case 790: S790(); break;
}
```

Pseudocode:

```
// use binary search to locate case
if (i <= 303) { // This eliminates half of the choices.
    if (i <= 10) {
        if (i == 0) {
            S0();
        else if (i == 10) {
            S10();
        }
    } else {
        ...
...
```

When you use switch, compiler automatically chooses how to speed it up.