Flow of Control:

- Programs can control the order in which their instructions are executed.
- Four types of flow:
  1. Sequential:
     - Execute instructions in the order listed in the code.
  2. Method calls:
     - Transfer flow control to the code inside the method;
     - Control returns back to the point of the call. Some calls also return a value.
  3. Selection:
     - Which set of instructions are executed depends on the data.
  4. Looping:
     - Repeat a set of instructions, changing some of the data each time through the set of instructions.

<u>Comparison Operators</u>:

- Equality operators are the same as in Python
  - **`==`** compares two values and returns **`true`** if one is equal to the other.
  - **`!=`** compares two values and returns **`true`** if one is <u>not</u> equal to the other.
- But what does it mean for things to be equal in Java?
- It means the value stored at that location is equal.
- For primitive types it means they have the same value

```
int i = 5;
int j = 9 - 4;
   i == j // evaluates to true
```

- For reference variables (objects) it means they refer to the same object

```
String a = "one";
String b = a;
String c = "on" + 'e';
       a == b // evaluates to true
       a == c // evaluates to false if though both have "one"
```

Equality Operators (continued):

- Do not confuse these two:
  - **`==`** equality operator.
  - **`=`** assignment operator.
- Examples:

```
int aNum = 42, bNum = 96;
boolean answer;
answer = (aNum = bNum);
```

Gives the error:
“Incompatible types”.
Why?

```
boolean ans1 = true, ans2 = false;
Boolean answer;
answer = (ans2 = ans1);
System.out.println("answer is " + answer);
```

Compiles without error.
Runs without error.
What is printed?

Relational Operators:

- Result is boolean
  - **`true`** or **`false`**

Relational Operators	Type	Meaning
`<`	binary two operands	is less than
`<=`	binary two operands	is less than or equal to
`>`	binary two operands	is greater than
`>=`	binary two operands	is greater than or equal to

<u>**Logical Operators**</u>:

- The **`!`** operator:
  - Performs a NOT operation. (same as Python **`not`**)
  - Has only <u>one</u> operand.
  - Returns **`false`** if the operand is **`true`**. Returns **`true`** if the operand is **`false`**.
  - Example:

```
float income;
income = inputScan.nextFloat();

boolean rich, poor;

rich = (income > 1e5);

poor = !(income > 1e5);

poor = !rich;
```

Both of these give the same result.

<u>Logical Operators</u> (continued):

- The **`&&`** operator:
    - Performs an AND operation (same as Python **`and`**).
    - Has two operands.
    - Returns **`true`** if <u>both</u> operands are **`true`**; otherwise, **`false`** is returned.
    - Example:

```
int age;
age = inputScan.nextInt();

boolean teenAge;

teenAge = (age >= 13) && (age <= 19);

System.out.println("teenAge is " + teenAge);
```

- **The assignment could have also been written as:**

```
teenAge = (age > 12) && (age < 20);
```

Logical Operators (continued):

- The `||` operator:
  - Performs an OR operation (same as Python `or`)
  - Has two operands.
  - Returns **`true`** if either or both operands are **`true`**; otherwise, **`false`** is returned.
  - Returns **`false`** if both operands are **`false`**; otherwise, **`true`** is returned.
  - Example:

Two ways to think about what OR does.

```
int myAge;
myAge = inputScan.nextInt();

int sisterAge = 34, brotherAge = 39;

boolean notYoungest;

notYoungest = (myAge > brotherAge) || (myAge > sisterAge);

System.out.println("notYoungest is " + notYoungest);
```

- Summary in the form of a *truth table*:

`a`	`b`	`!a`	`a && b`	`a || b`
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

- Suppose we want to know if a value falls within a range.
- In Python we can write the following:

```
isLiquid = 0 <= waterTemp <= 100
```

  - This does not work in Java
- The following Java program is incorrect:

```
float waterTemp;
waterTemp = inputScan.nextFloat();
boolean isLiquid;
isLiquid = 0.0F <= waterTemp <= 100.0F;
```

  - The last line produces the error:
    "`operator <= cannot be applied to boolean,float`"
  - Why?

- What is the right way to make this range comparison?

```
liquidWater = (0.0F <= waterTemp) && (waterTemp <= 100.0F);
                   Boolean          and        Boolean
```

Comparing Floats and Doubles:

- Can compare floats and doubles using **`<`**, **`<=`**, **`>`**, **`>=`**, **`==`**, and **`!=`**.
- But, equality is a problem. Consider

```
double zap, wobble;
wobble = 1.1;
zap = 0.1;
zap = zap + 0.1;    // repeat this line 10 times
```

  - The two should now both be **`1.1`**, but **`zap`** is not!

```
wobble = 1.1
zap = 1.0999999999999999
```

  - An equality, or inequality, test will yield an unexpected result!
    - A less-than test or a greater-than test will also lead to an incorrect result.

<u>**Comparing Floats and Doubles**</u> (continued):

- Establish a "close enough" criteria.
  - How close together do the values need to be to be considered "equal"?
  - Example: choose **`0.001`** as being "close enough".
  - Find the difference between **`zap`** and **`wobble`**. Is this difference less than **`0.001`**?

```
double zap, wobble;
wobble = 1.1;
zap = 0.1;
zap = zap + 0.1;    // repeat this line 10 times
if ( Math.abs(zap - wobble) < 0.001 )
  System.out.println("zap and wobble are equal (close enough)");
else
  System.out.println("zap and wobble are not equal");
```

`if` Statements:

Simple if:

- Python:

```
if <condition-goes-here>:
   indented code
   indented code
code executed after the if statement
```

- Java:

```
if ( condition-goes-here ) {
   // true block
   // code to execute when condition is true
}
// code here that executes after the if statement
```

- Note the parenthesis around the condition are required!

`if` Statements:

Simple if (cont):

- The following is written in Python. How would you write it in Java?

```
if x == 10:
   done = True
```

- Java:

```
if (x == 10) {
   done = true;
}
```

```
if (x == 10) {
   done = true;
}
```

- Note the {} are optional if there is only a single statement. The following also works:

```
if (x == 10)
   done = true;
```

- Since line breaks don't matter, we can put all on one line

```
if (x == 10) done = true;
```

Warning:

- Leaving out the {} can be dangerous.

```
if (x == 10)
   done = True;
   System.out.println("I'm done with this!");
```

- The code above will print out

```
I'm done with this!
```

- even if **`x == 10`** is true. Java does not care about the indentation. Because the statements are not wrapped in a block ({ }), only the first statement is under the if.

if/else:

- Python

```
if <condition>:
   code
   code
elif <condition>:
   code
   code
elif <condition>:
   code
   code
else
   code
   code
code out of if block
```

- Java

```
if (<condition>) {
   code
   code
} else if (<condition>) {
   code
   code
} else if (<condition>) {
   code
   code
} else {
   code
   code
}
code out of if block
```

example

- Python

```
if score >= 90:
   grade = 'A'
elif score >= 80:
   grade = 'B'
elif score >= 70:
   grade = 'C'
elif score >= 60:
   grade = 'D'
else
   grade = 'E'
print(grade)
```

- Java

```
if (score >= 90) {
   grade = 'A';
} else if (score >= 80) {
   grade = 'B';
} else if (score >= 70) {
   grade = 'C';
} else if (score >= 60) {
   grade = 'D';
} else {
   grade = 'E';
}
System.out.println(grade);
```

- Warning, this code does **<u>not</u>** work.

```
if ( waterTemp <= 0 )
   if ( waterTemp <= -10 )
      System.out.println("Ice skating time!");
else if ( waterTemp <= 18 ) {
   System.out.println("Go for a swim!");
   System.out.println("Bring a wet suit!");
} else if ( waterTemp <= 37 ) {
   System.out.println("Go for a swim!");
} else {
   System.out.println("Hot tub time!");
   System.out.println("Don't stay in too long!");
}

System.out.println("Have a good time!");
```

- When looking at an **`else`**, how does the compiler know what **`if`** it belongs to?
  - Recall: The compiler ignores indentation!
  - Each **`else`** is associated with the most recent **`if`** that does not already have an **`else`**.

```
if ( waterTemp <= 0 )
   if ( waterTemp <= -10 )
      System.out.println("Ice skating time!");
else if ( waterTemp <= 18 ) {
   System.out.println("Go for a swim!");
   System.out.println("Bring a wet suit!");
} else if ( waterTemp <= 37 ) {
   System.out.println("Go for a swim!");
} else {
   System.out.println("Hot tub time!");
   System.out.println("Don't stay in too long!");
}

System.out.println("Have a good time!");
```

- Use **{ }**'s to surround the **if** statement that does not have an **else**.

```
if ( waterTemp <= 0 ) {
     if ( waterTemp <= -10 )
        System.out.println("Ice skating time!");
   } else if ( waterTemp <= 18 ) {
     System.out.println("Go for a swim!");
     System.out.println("Bring a wet suit!");
   } else if ( waterTemp <= 37 ) {
     System.out.println("Go for a swim!");
   } else {
     System.out.println("Hot tub time!");
     System.out.println("Don't stay in too long!");
   }

   System.out.println("Have a good time!");
```

switch:

- An **if/else if** statement can (sometimes) be replaced by a **switch** statement.
- Requirements:
  - Must be comparing the value of a **char**, **byte**, **short**, or **int**.
  - Note: cannot be a **long**, **float**, **double**, **String**, or anything else!

```
switch ( /* char, byte, short, or int expression goes here */) {
   case constant1:
      // statement(s);
      break;  // optional
   case constant2:
      // statement(s);
      break;  // optional
   ...
   default:   // optional (but generally a very good idea!)
      statements(s);
} // switch ends here
```

The **switch** block

switch (continued):

- The *expression* is evaluated, then its value is compared to the **`case`** constants in order.
- When a match is found, the statements under that **`case`** constant are executed in sequence until:
  - a **`break`** statement is reached, OR
  - the end of the **`switch`** block is reached.
- Example:
  - A program that reads a year from the keyboard.
  - Determines if the year is:
    - A Presidential election year.
    - A House of Representatives year.
    - A year with no federal election.

```
Scanner inputScan = new Scanner( System.in );
short year;
System.out.print("Enter the year: ");
year = inputScan.nextShort();

switch ( year % 4 ) {
   case 0:    //  if ( year % 4 == 0 )
      System.out.println("Elect a President");
      System.out.println("Elect members of the US House");
      break;

   case 2:
      System.out.println("Elect members of the US House");
      break;

   default:
      System.out.println("No federal election");
      break;
}
```

<u>switch</u> (continued):

- A simplistic class standing example:
  - Students "advance" from freshman to sophomore, etc. every 30 credit hours.
    - Freshmen can only take freshman classes.
    - Sophomores can take sophomore and freshman classes.
    - Etc.
  - Once they reach 120 credit hours, they "advance" to graduate status.
    - From 120 to 134 credit hours, they can take 500-level courses.
    - From 135 to 150 credit hours, they can take 500- and 600-level courses.
  - Number of credit hours beyond 150 (and below 0) are not allowed.

<u>switch</u> (continued) — **`ClassStanding`** example continued.

- Can use the “fall through” feature of switch.
- This bit of code handles undergrads. What gets printed if the creditHours is 91?

```
switch (creditHours / 30) {
     case 3:
        System.out.println("Can take Senior courses");

     case 2:
        System.out.println("Can take Junior courses");

     case 1:
        System.out.println("Can take Sophomore courses");

     case 0:
        System.out.println("Can take Freshman courses");
        break;
}
```

switch (continued) — `ClassStanding` example continued.

- Graduate students are in the range 120 to 149.
  - This group can be found by integer division by 30.
  - There are two sub-categories, note, graduate students may not take undergraduate courses.

```
switch (creditHours / 30) {
     case 4:
        if ( creditHours >= 135 ) {
           System.out.println("Can take 600-level courses");
        }
        System.out.println("Can take 500-level courses");
        break;
     case 3:
        System.out.println("Can take Senior courses");
     case 2:
        System.out.println("Can take Junior courses");
     case 1:
        System.out.println("Can take Sophomore courses");
     case 0:
        System.out.println("Can take Freshman courses");
        break;
}
```

switch (continued) — **`ClassStanding`** example continued.

- How to handle **`creditHours`** that are negative or too large?

```
switch (creditHours / 30) {
    case 4:
       if ( creditHours >= 135 ) {
          System.out.println("Can take 600-level courses");
       }
       System.out.println("Can take 500-level courses");
       break;
    case 3:
       System.out.println("Can take Senior courses");
    case 2:
       System.out.println("Can take Junior courses");
    case 1:
       System.out.println("Can take Sophomore courses");
    case 0:
       System.out.println("Can take Freshman courses");
       break;
    default:
       System.out.println("The credit hours must be 0 to 149");
}
```

Conditional Operator:

- The conditional operator contributes one of two values to an expression based on the value of the condition.
- Syntax:

```
          condition    ? trueExp : falseExp
     kilo = (hotel < golf) ?    42   :   -378;
```

- Has the same meaning as:

```
   if ( hotel < golf )
        kilo = 42;
     else
        kilo = -378;
```

- The conditional operator `?:` is a *ternary* operator in that it requires 3 operands: *condition*, *trueExp*, *falseExp*.
  - It is the only ternary operator in Java.

Conditional Operator (continued):

- Want to print a message that uses the correct singular or plural form:
  - Without the conditional operator, we can write:

```
if ( numBoxes == 1 )
      System.out.println("We need 1 box.");
  else

      System.out.println("We need " + numBoxes + " boxes.");
```

  - With the conditional operator, we can write:

```
 System.out.println("We need " + numBoxes +
                         ((numBoxes == 1) ? " box." : " boxes."));
```