CS201 Introduction to Java Programming
Control Statements

Chengyu Sun
California State University, Los Angeles

Overview

- Branch
  - if
  - if … else
  - switch
- Loop
  - while
  - do … while
  - for
- Break and continue
  - break
  - continue

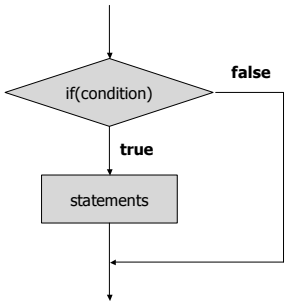
if

```
if( condition )
{
   statements
}
```

```
if( condition ) statement;
```

- If the condition is true, execute the statement(s)

Flow Chart of if Statement



Example – Large and Small

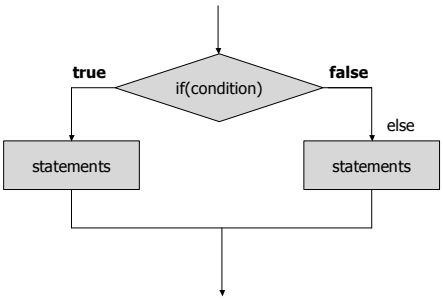
- Input two integers, and store the larger of the two in the variable `large` and the other one in the variable `small`

if … else

```
if( condition )
{
   some statements
}
else
{
   some other statements
}
```

- Why do we need `if … else` when we have `if`?

Flow Chart of if … else Statement



Example: Score to Grade

- Rewrite the program ScoreToGrade using `if … else` statement

Nested if … else

```
// Score to Grade

if( score >= 90 )
{
   System.out.println( "A" );
}
else
{
   if( score >= 80 )
   {
      System.out.println( "B" );
   }
   else
   {
      System.out.println( "C" );
   }
}
```

```
// Score to Grade

if( score >= 90 )
{
   System.out.println( "A" );
}
else if( score >= 80 )
{
   System.out.println( "B" );
}
else
{
   System.out.println( "C" );
}
```

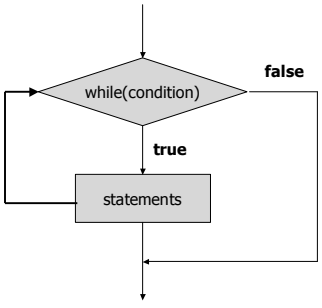
while

```
while( condition )
{
   statements
}
```

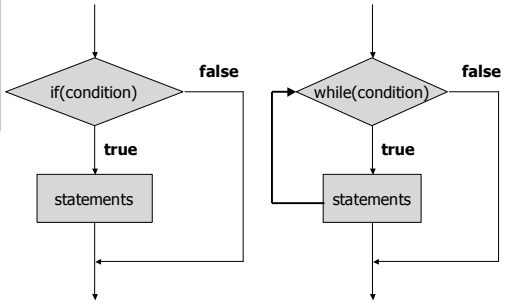
```
while( condition ) statement;
```

- If the condition is true, execute the statement(s); repeats until the condition is no longer true

Flow Chart of while Statement



if vs. while



Example: Sum100

- Calculate 1 + 2 + 3 + … + 99 + 100

Example: Average of N Numbers

- Calculate the average of n given numbers

do … while

```
do
{
   statements

} while ( condition );
```

- The statements will be executed at least once.
- Flow chart ??

Example: Sum100 Revisited

- Rewrite the program Sum100 using do…while statement

for

```
for( initialization ; loop-condition ; increment )
{
   // do something
}
```

is equivalent to:

```
initialization;
while( loop-condition )
{
   // do something
   increment;
}
```

for loop example

```
// Sum100 using for loop
int sum = 0;
for( int number=1 ;
      number <= 100 ;
      ++number )
{
   sum += number;
}
```

```
// Sum100 using while loop
int sum = 0;
int number = 1;
while( number <= 100 )
{
   sum += number;
   ++number;
}
```

break

- breaks out of a loop or switch statement

```
// Sum100 using while and break
int sum = 0;
int number = 1;
while( true )
{
   sum += number;
   ++number;
   if( ?? ) break;
}
```

continue

- Skip the rest of the current iteration
- E.g. calculate the sum of 1 + 2 + … + 49 + 51 + … + 100
- Exercise: what happens if we replace *continue* with *break* in the example?

```
// Sum100 – 50
int sum = 0;
int number = 1;
while( number <= 100 ) {
   if( number == 50 )
   {
      ++number;
      continue;
   }
   sum += number;
   ++number;
}
```

switch

```
switch( integer expression )
{
   case value1:
      // do something

   case value2:
      // do something
   …

   default:
      // do something
}
```

- Integer expression
  - An integer variable, or
  - An integer value, or
  - Some statement that produces an integer
- Integer types for *switch*
  - byte – 8 bit
  - char – 16 bit
  - short – 16 bit
  - int – 32 bit

switch example

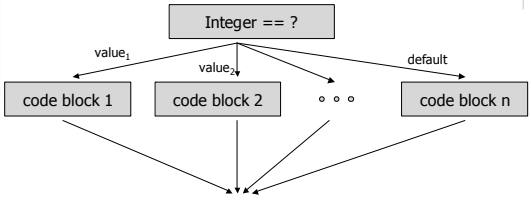
```
// day is M, T, W, R, F, S

switch( day )
{
   case 'M':
      System.out.println("Monday");
      break;

   case 'T':
      System.out.println("Tuesday");
      break;

   case 'W':
      System.out.println("Wednesday");
      break;

   case 'R':
      System.out.println("Thursday");
      break;

   case 'F':
      System.out.println("Friday");
      break;

   case 'S':
      System.out.println("Saturday");
      break;

   default:
      System.out.println("Error!");
}
```

More about switch

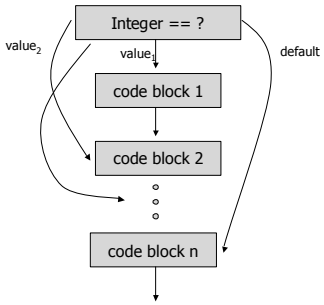
- Why *break*?
- No *break*
- *default* is optional, but it's usually nice to have it

```
// day is M, T, W, R, F, S

switch( day )
{
   case 'm':
   case 'M':
      System.out.println("Monday");
      break;

   case 't':
   case 'T':
      System.out.println("Tuesday");
      break;

   …
}
```

Control Flow of switch – Conceptual



Control Flow of switch – Actual



?: operator

boolean expression ? *op1* : *op2*;

- returns *op1* if the *boolean expression* evaluates to *true*; otherwise return *op2*
- Generally, a shorthand for *if … else* statement, but
- Not always
- Fairly low precedence
- Right-associative

?: examples

```
max = a > b ? a : b;
max = a < b ? b : a;
min = a > b ? b : a;
min = a < b ? a : b;

a > b ? max=a : max=b; // ??
10 > 3 ? 11+12 > 22 : 11+12 < 22; // ??
10 > 3 ? 11+12 > 22 : 22; // ??

true ? true ? true : false : false ? false : true; // ??
false ? false ? false : true : true ? true : false; // ??

score >= 90 ? 'A' : score >= 75 ? 'B' : 'C'; // ??
```