Lecture Set #11:
Ternary Operator, Switch, Break, Continue

1. ternary operator: The ?: (conditional operator)
2. switch
3. break/continue

The Conditional Operator

• The only ternary operator (has 3 operands)
• Format:
  + boolean-expression?expression1:expression2
• Purpose:
  + test to see if boolean-expression is true or false
  + whole expression takes on the value of expression1 when boolean-expression was true
  + whole expression takes on the value of expression2 when boolean-expression was true
• See examples

What is another way to write this if-else-if statement?

```java
if (grade == 'A')
    System.out.println("I'm very happy");
else if (grade == 'B')
    System.out.println("I'm relatively happy");
else if (grade == 'C')
    System.out.println("At least I get credit");
else
    System.out.println("Check with the professor");
```
The switch Statement: General Form

switch (control-expression) {
  case case-label-1 :
    statement-sequence-1 break;
  case case-label-2 :
    statement-sequence-2 break;
  …
  case case-label-n :
    statement-sequence-n break;
  default:
    default-statement-sequence break;
}

Our text says it cannot be a byte or short. This is wrong!
The control-expression is one of the following types:
  char, int, short, byte
Each case label must be a value in type of control expression.
You may have any number of statements, including if-else and loops.
The "break" statement jumps out of the switch statement.
The optional "default" case is executed if no other case matches.

The default Case

- default is optional
  If omitted, and no case matches, then the switch statement does nothing
- However, you should always include a default case, even if you want nothing to be done if no case matches (you should never rely on implicit behavior!)
- Although cases are not required to be in order... (following is legal):
  
  switch (option) {
    case 2:
    …
    case 9:
    …
    default:
    …
    case 1:
    …
  }

  It is much better to list cases:
  - in increasing order
  - with default last

Case Continuation

- The control expression can have one of the following types:
  char, int, short, byte
- not float, double, boolean, long
- not a String or other object
- Case continuation also called "cascading case behavior", "falling through to the next case", etc.
- It is occasionally handy for combining of cases
  e.g. case-insensitivity
  switch (grade) {
    case 'a':
    case 'A':
      System.out.println("I'm very happy");
      break;
  }

- Be very careful about using this cascading behavior!
  - Always insert break statements after every case
  - Then remove ones you do not want

Our text says it cannot be a byte or short. This is wrong!

Each case label must be a value in type of control expression.
You may have any number of statements, including if-else and loops.
The "break" statement jumps out of the switch statement.
The optional "default" case is executed if no other case matches.
Why Use switch?

- switch can also be implemented using if-else
- switch also restricted in terms of data types in control statements
- Including break statements is a pain
- However
  - switch often more efficient (compiler generates better code)
  - Code can be more compact because of case-continuation behavior
  - Sometimes case analysis is clearer using switch

More about break for loops

- break can also be used to exit immediately from any loop
  - while
  - do-while
  - for
- e.g. “Read numbers from input until negative number encountered”
  Scanner sc = new Scanner (System.in);
  int n;
  while (true) {
    n = sc.nextInt ();
    if (n < 0)
      break;
    else
      // process n;
  }
  Loop only terminates when break executed
  This only happens when n < 0

Warning about break

- Undisciplined use of break can make loops impossible to understand
  - Termination of loops without break can be understood purely by looking while, for parts
  - When break included, arbitrary termination behavior can be introduced
- Rule of thumb: use break only when loop condition is always true (i.e. break is only way to terminate loop)
- When you use it, make sure it has a good comment explaining what is happening
continue Statement

- `continue` can also be used to affect loops
- `break` halts loops
- `continue` jumps to bottom of loop body
- Following prints even numbers between 0 and 10
  ```java
  for (int i = 0; i <= 10; i++)
      if (i % 2 == 1)
          continue;
      System.out.println (i);
  ```
- Effect of `continue` statement is to jump to bottom of loop immediately when `i` is odd
- This bypasses `println`
- `continue` should be avoided
- Confusing
- Easy equivalents exist (e.g. `if-else`)
- Included in Java mainly for historical reasons
- When you use it, make sure it has a good comment explaining what is happening