Lecture Set #11: Ternary Operator and Switch

- Method Overloading Warning
- ternary operator: The ?: (conditional operator)
- switch

Method Overloading

- Method definition
  ```java
  public static void f(int x, float y){
      body
  }
  ```
- prototype:
  ```java
  public static void f(int x, float y)
  ```
- signature:
  ```java
  f(int, float )
  ```
- You can only overload methods if they have different signatures.
- Implicit widening conversions are allowed
  - Beware of subtle problems with widening conversions

The Conditional Operator

- The only ternary operator (has 3 operands)
- Format:
  ```java
  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 false
- 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

```java
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;
}
```

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 not rely on implicit behavior!)
- Although cases are not required to be in order ... (following is legal):
  ```java
  switch (option) {
    case 2:
    case 3;
    default:
    case 1:
  }
  ```
- It is much better to list cases:
  - in increasing order
  - with default last

Our text says it cannot be a byte or short. This is wrong!
The optional "default" case is executed if no other case matches.
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
... 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

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