Decisions: the switch statement

Pieter Joubert

October 8, 2023

Decisions: the switch statement

Table of Contents

Section 1

Making Accurate and Efficient Decisions

Range checks

- There are a number of ways to insure that you are performing your decisions accurately and efficiently.
- One common problem that we use if statements to solve is a range check, i.e. checking if a value is within a certain set of ranges, e.g. 0 to 10, 11 to 50, 51 to 100.

```
if(charge >= 100) {
    damage += 10;
}
if(charge >= 50) {
    damage += 5;
}
if(charge >= 10) {
    damage += 2;
}
```

Range checks: fixed

- As we can see there are some problems with the previous range check code.
- Not only is it inefficient (i.e. it will always do 3 checks), it is inaccurate: a high charge could still result in a low damage.
- Let's look at one possible better solution.

```
if(charge >= 100) {
     damage += 10;
} else {
     if(charge >= 50) {
          damage += 5;
     } else {
          damage += 2;
     }
}
```

Section 2

Using switch

Multiple *if* statements

- Very often we need to program a large number of decisions in our code.
- One naive way of approaching this is to have a large selection of if statements.
- While this will work it can very easily get cumbersome and become difficult to read:

Many ifs - Example

```
if(keyCode == "a") {
    moveLeft();
}
if(keyCode == "d") {
    moveRight();
}
if(keyCode == "w") {
    moveUp();
}
if(keyCode == "s") {
    moveDown();
}
if(keyCode == "space") {
    jump();
}
if(keyCode == "e") {
    interact();
}
```

Using the switch statement

- Instead we can use the *switch* statement.
- The switch statement tests one variable against multiple conditions, and then executes code based on each possible condition.

```
switch(keyCode) {
        case "a"  :  moveLeft();
        case "d"  :  moveRight();
        case "w"  :  moveUp();
        case "s"  :  moveDown();
        case "space"  :  jump();
        case "e"  :  interact();
}
```

Section 3

Using the Conditional and NOT operators

The conditional operator

- The conditional operator is a *abbreviated* form of the if statement.
- The format of the conditional operator is as follows: *testExpression*?*trueResult* : *falseResult*;
- We define what expression we want to test before the question mark.
- We define what the result will be if the expression is true between the question mark and the colon.
- We fine what the result will be if the expression is false after the colon.

Conditional operator - example

```
//keeping health above or equal to zero
health = (health - damage > 0) ? health - damage : 0;
```

The NOT operator

- We use the NOT operator to negate the result of a Boolean expression.
- The NOT operator is the exclamation mark: !
- We can also use the NOT EQUAL TO operator: !=

```java
boolean dead = false;
if(!dead) {
    System.out.println("I'm alive");
}

if(dead != true) {
    System.out.println("I'm alive");
}
```

Section 4

Understanding Operator Precedence

Operator Precedence

- The operators we've been looking at in this class also have an order of precedence like the arithmetic operators.
- We can actually combine arithmetic operators with boolean operators in the same expression.
- For this reason it is very important to understand the order of precedence to ensure your expression is being evaluated correctly.
- The next slide shows the full order of precedence.
- Always consider wrapping an expression in parentheses to make the order of operations explicit to the compiler as wel as clearer to the reader.

Operator Precedence

PRECEDENCE	OPERATOR
Highest (9)	!
8	* / %
7	+ −
6	> < >= <=
5	== ! =
4	&&
3	||
2	? :
1	=

Section 5

Example of decisions in Constructors

Decisions in Constructors

- Let’s look at a code example of using decisions in constructors.

Section 6

Lecture summary

Lecture summary

- Planning Decision Making Logic
- if and if-else statements
- Using multiple if and if-else statements
- Nesting if and if-else statements
- Using Logical AND and OR operators

Questions

Thank you! Questions?