Making Decisions

Lecture 5 - B

Object-Oriented Programming

Agenda

- Blocks
- Statements
- Selection Statement
  - If-else Statement
  - Nested if / if-else Statements
  - Switch Statement
- Relational Operators
- Logical Operators
- Common Errors

Blocks (1)

- Question
  - Why we use curly braces { } in our classes, methods, etc?
- Answer
  - Curly braces { } is the simplest type of structure a program can provide.
  - It allows the compiler to do memory management of the executing program.
  - Anything declared inside a pair of curly braces is a self-contained entity.

Blocks (2)

```
{
   int temp;
   temp * 2;
   System.out.println(temp)
}
  System.out.println(temp)
```

- The statements inside the curly braces is a self-contained code.
- temp will be removed from the memory at the end of the ending brace.
- The last line will give an error because no temp variable is available at this place in the memory.
- Blocks are used with classes, methods and many other constructs. Some of which we are going to study today.

Statements

- So far we have studied code that execute only once.
- This code also executes in a sequence.
- The execution of a program trace is called its flow-of-control.
- What if we need some statements to execute depending on one condition and others depending on another statement.

$$\begin{tikzcd}\boxed{\text{Statement 1}} \arrow[d] \\ \boxed{\text{Statement 2}} \arrow[d] \\ \boxed{\text{Statement 3}} \arrow[d] \\ \boxed{\text{Statement 4}}\end{tikzcd}$$

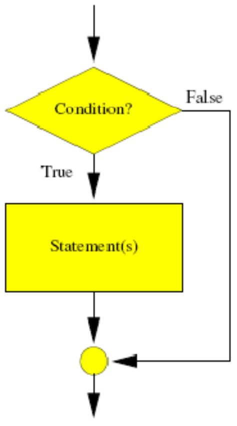
Selection Statement

- Choice between different actions during execution
- Choice is based on a criterion specified during the selection statement
  - criteria : boolean condition (**`if, if-else`**)
    - Choice:
      - "if it rains I will go shopping" (`if`)
      - "if it rains I will go shopping, otherwise I will go to the park" (`if-else`)
  - criteria : a specific value among a set of values (**`switch`**)
    - Example:
      - On "Monday", I do my shopping
      - On "Tuesday", I work
      - On "Wednesday", I go to the library
      - On "Thursday", I go to the swimming pool
      - Etc.

Simple `if` Statement

```
if ( <conditional expression> )
     <statement>


if ( <conditional expression> ){
     <statement_1>
    <statement_2>
       …
    <statement_n>
}
```

Simple `if` statement



Simple `if` Statement

```
if ( <conditional expression> )
      <statement>
```

Condition must evaluate to true

```
public void aSimpleCondition(double price, double money)
{
        if (money >= price)
                System.out.println("Payment accepted");

}
```

End of if statement

Simple **`if`** Statement

```
if ( <conditional expression> )
{
      <statement_1>
      <statement_2>
         …
      <statement_n>
}
```

Simple `if` Statement

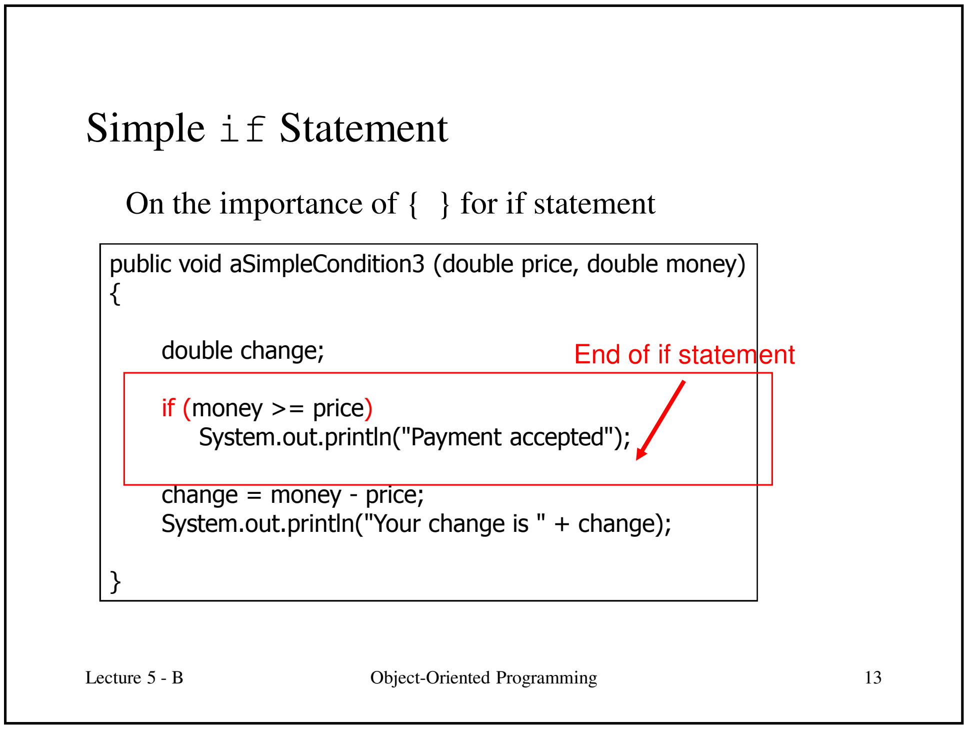
```
public void aSimpleCondition2 (double price, double money)
{

      double change;

      if (money >= price) {
          System.out.println("Payment accepted");
          change = money - price;
          System.out.println("Your change is " + change);
      }
}
```

End of if statement

Simple `if` Statement

On the importance of { } for if statement

```
public void aSimpleCondition3 (double price, double money)
{

      double change;

      if (money >= price)
          System.out.println("Payment accepted");
          change = money - price;
          System.out.println("Your change is " + change);

}
```



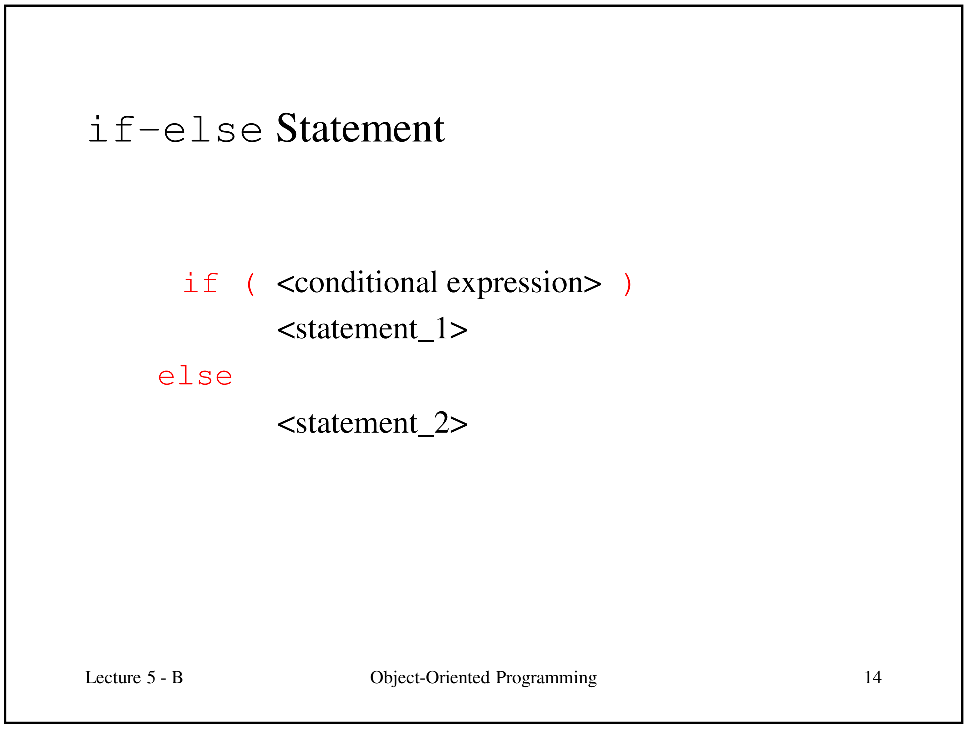
Simple `if` Statement

On the importance of { } for if statement

```
public void aSimpleCondition3 (double price, double money)
{

    double change;

    if (money >= price)
        System.out.println("Payment accepted");
    change = money - price;
    System.out.println("Your change is " + change);

}
```

End of if statement

Lecture 5 - B   Object-Oriented Programming   13



`if-else` Statement

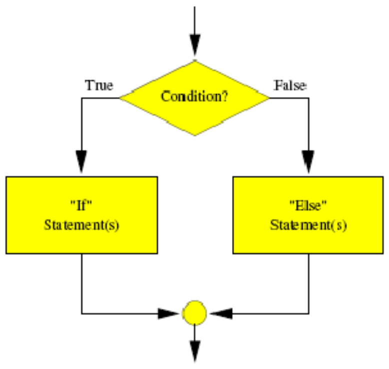
```
     if ( <conditional expression> )
          <statement_1>
    else
          <statement_2>
```

Lecture 5 - B   Object-Oriented Programming   14

`if-else` Statement

```
if ( <conditional expression> )
{
        <statement_1>
        …
        <statement_n>
}
else
{
        <statement_o>
        …
        <statement_z>
}
```

`if-else` Statement



`if-else` Statement

Condition evaluates to true

```
public void aSimpleConditionWithElse(double price, double money)
{
       if (money >= price)
           System.out.println("Payment accepted");
       else
           System.out.println("Payment not accepted");

 }
```

Condition evaluates to false

`if-else` Statement

```
public void aSimpleConditionwithElse2(double price, double money)
{
      if (money >= price) {
          double change;
          System.out.println("Payment accepted");
          change = money - price;
          System.out.println("Your change is " + change);
      }
      else {
         double missingMoney;
         missingMoney = price - money;
         System.out.println("Payment not accepted");
         System.out.println("Please complete with " + missingMoney);
       }
}
```

Nested if / if-else Statements

```
public void nesteIfA(int x, int y)
{
        if ( x > 0 )
        {
          if (y > 0)
                System.out.println("Both x and y are positive");
        }
        else
          System.out.println("x is negative, y may be positive or negative");

}
```

End of second if statement

Nested if / if-else Statements

```
public void nesteIfC (int x, int y)
{
    if ( x > 0 )
          if (y > 0)
            System.out.println("Both x and y are positive");
    else
       System.out.println("x is negative, y may be positive or negative");

}
```

Be careful with the use of { }

Changes the tested values

Nested `if` / `if-else` Statements

```
public void nesteIfB(int x, int y)
{
      if ( x > 0 )
             if (y > 0)
               System.out.println("Both x and y are positive");
             else
                System.out.println("x is positive and y is negative");

}
```

Be careful to the use of { }

Changes the tested values

Nested `if` / `if-else` Statements

```
public void nestedIf2 (int testscore) {
      char grade;

      if (testscore >= 90) {
         grade = 'A';
      } else if (testscore >= 80) {
               grade = 'B';
             } else if (testscore >= 70) {
                      grade = 'C';
                    } else if (testscore >= 60) {
                             grade = 'D';
                           } else grade = 'F';

       System.out.println("Grade = " + grade);
 }
```

switch Statement

```
switch ( <integral expression> ) {
     case label_1:
            <statement_1>
     case label_2:
              <statement_2>
     …
     case label_n:
            <statement_n>
     default:
           <statement>
}
```

Evaluated First

Value of integral expr compared with first case label

Value of integral expr compared with second case label

Continues until the end unless “break”

switch Statement

```
switch ( <integral expression> ) {
     case label_1:
            <statement_1>
     case label_2:
              <statement_2>
     …
     case label_n:
            <statement_n>
     default:
           <statement>
}
```

char
byte
short
int
(enumerated type)

`switch` Statement

```
public void switchExample(char digit) {

      switch (digit) {
         case '1': System.out.println("First "); break;
         case '2': System.out.println("Second ");break;
         case '3': System.out.println("Third ");break;
         case '4': System.out.println("Fourth ");break;
         case '5': System.out.println("Fifth ");break;
         default: System.out.println("Other ");
      }
   }
```

`switch` Statement

```
public void switchExample2(int digit) {

      switch (digit) {
         case 1: System.out.println("First ");
         case 2: System.out.println("Second ");
         case 3: System.out.println("Third ");
         case 4: System.out.println("Fourth ");
         case 5: System.out.println("Fifth ");
         default: System.out.println("Other ");
      }
}
```

If no break, all statements after condition are satisfied until the end of the switch statement

Relational Operators

Operator	Meaning
==	is equal to
>=	is greater than or equal to
<=	is less than or equal to
<	is less than
>	is greater than
!=	is not equal to

- Numerical expressions (variables, constants, literal constants, etc…) can be compared using these operators. They return a `boolean` value: either `true` or `false`
- Note that `==` is used to test for equality; don't confuse it with `=` of assignment!

```
boolean isEven = ((x % 2) == 0);
```

Logical Operators

- And (`&&`) takes two boolean expressions and returns true only if both expressions are true
- Or (`||`) takes two boolean expressions and returns true if at least one is true
- Not (`!`) takes one boolean expression and returns its negation
- Examples:

```
boolean bool1 = (3 == 2) && (2 < 3); // false
boolean bool2 = (!bool1) || (5.6 >= 8); // true
boolean bool3 = !(bool1 && bool2); // true
```

Common Errors

- Compound Conditional Error

  `value >= 0 || value <= 10`

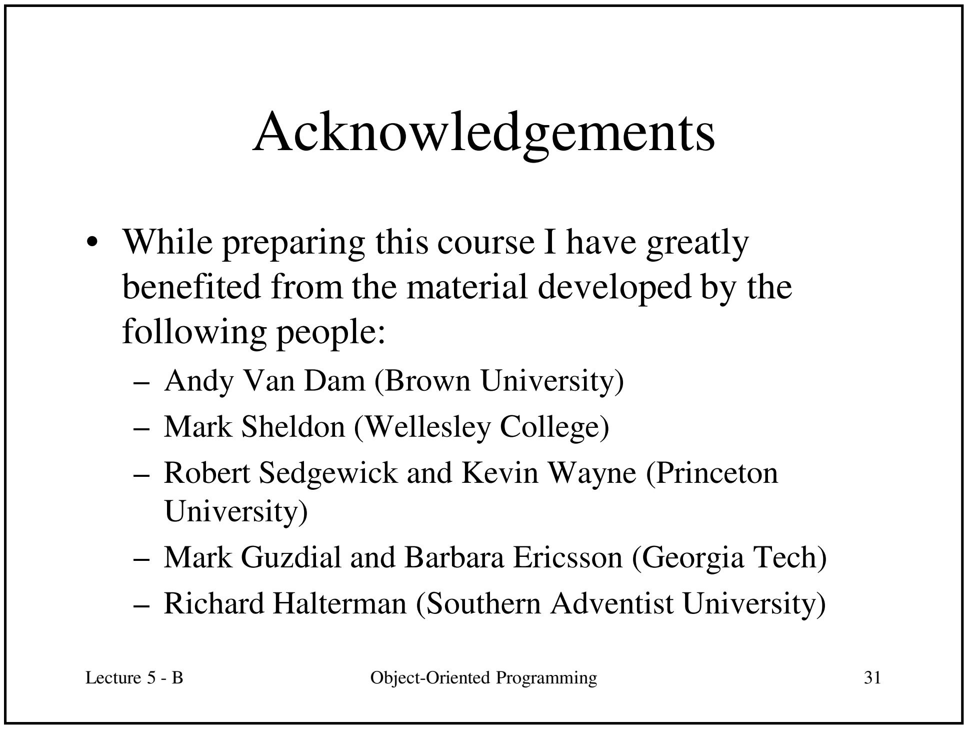
  – This expression is a *tautology.* A tautology is a boolean expression that is always true.
- Contriving Compound Boolean Expression

  ```
  – If (value < 0 && value > 10) {
     //some code
     }
  ```

  – This condition can never be true. A number cannot be both be less than zero and greater than ten.

Readings

Book Name: Big Java
Author: Cay Horstmann
Content: Chapter # 5



Acknowledgements

- While preparing this course I have greatly benefited from the material developed by the following people:
  - Andy Van Dam (Brown University)
  - Mark Sheldon (Wellesley College)
  - Robert Sedgewick and Kevin Wayne (Princeton University)
  - Mark Guzdial and Barbara Ericsson (Georgia Tech)
  - Richard Halterman (Southern Adventist University)