Re: Primitive Numerical Data Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>$-2^7$ (-128) to $2^7-1$ (127)</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>$-2^{15}$ (-32768) to $2^{15}-1$ (32767)</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int</td>
<td>$-2^{31}$ (-2147483648) to $2^{31}-1$ (2147483647)</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>long</td>
<td>$-2^{63}$ to $2^{63}-1$ (i.e., $-9223372036854775808$ to $9223372036854775807$)</td>
<td>64-bit signed</td>
</tr>
<tr>
<td>float</td>
<td>Negative range: $-3.4028235E+38$ to $-1.4E-45$</td>
<td>32-bit IEEE 754</td>
</tr>
<tr>
<td></td>
<td>Positive range: $1.4E-45$ to $3.4028235E+38$</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>Negative range: $-1.7976931348623157E+308$ to $-4.9E-324$</td>
<td>64-bit IEEE 754</td>
</tr>
<tr>
<td></td>
<td>Positive range: $4.9E-324$ to $1.7976931348623157E+308$</td>
<td></td>
</tr>
</tbody>
</table>
Objectives

° To declare `boolean` type and write Boolean expressions
° To use 6 comparison operators and 4 boolean operators
° To use Boolean expressions to control selection statements
° To implement selection control using `if` and nested `if` statements
° To implement selection control using `switch` statements
° To write expressions using the conditional operator
° To display formatted output using the `System.out.printf` method and to format strings using the `String.format` method
° To know the rules governing operator precedence and operand evaluation order

The `boolean` Type and Operators

° **Sequence**: a series of executable statements
° **Selection**: we often choose actions based on conditions (logical branching)
° **Conditions**: often in a program you need to compare two values, such as whether `i` is greater than `j`.
° Java provides six comparison operators (also known as relational operators) that can be used to compare two values.
° The result of the comparison is a Boolean value: true or false.

```
boolean b = (1 > 2);
```

The Boolean values CANNOT be cast to other types!
### Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
</tbody>
</table>

// a common error by “==”

* No space between those two signs (compile error)

### Boolean Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
</tr>
<tr>
<td>&amp; &amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or</td>
</tr>
</tbody>
</table>

* No space between those two signs (compile error)
Truth Tables for Operators

<table>
<thead>
<tr>
<th>p</th>
<th>!p</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 &amp; p2</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 ^ p2</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

Formulating Boolean Expressions: Examples

Expression to test if a numeric variable \(x\) is between values \(\text{min}\) and \(\text{max}\)

\((\text{min} \leq x) \&\& (x \leq \text{max})\)

Expression to test if \(x\) is outside the range \([\text{smallest}, \text{largest}]\)

\((x < \text{smallest}) \text{ or } (x > \text{largest})\)

Expression to test if \(x\) either equals 5 or is between 9 and 14

\((x == 5) \text{ or } ((9 \leq x) \&\& (x \leq 14))\)

Expression to test if \(x\) is not equal to 17 and not to 21 and is odd

\((x \neq 17) \&\& (x \neq 21) \&\& (x \% 2 == 1)\)

Expression to test if \(x\) is divisible by 2 and 3

\((x \% 2 == 0) \&\& (x \% 3 == 0)\)

Expression to test if a \(\text{char}\) variable \(ch\) is a letter

\((ch \geq 'A' \&\& ch \leq 'Z') \text{ or } (ch \geq 'a' \&\& ch \leq 'z')\)
Example: Determining Leap Year

This program first prompts the user to enter a year as an int value and checks if it is a leap year. A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

```java
import javax.swing.JOptionPane;
public class LeapYear {
    public static void main(String args[]) {
        // Prompt the user to enter a year
        String yearString = JOptionPane.showInputDialog("Enter a year");
        // Convert the string into an int value
        int year = Integer.parseInt(yearString);
        // Check if the year is a leap year
        boolean isLeapYear;
        isLeapYear = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
        // Display the result in a message dialog box
        JOptionPane.showMessageDialog(null, year + " is a leap year? " + isLeapYear);
    }
}
```

Operator Precedence

How to evaluate 3 + 4 * 4 > 5 * (4 + 3) - 1?

- +, - (Unary plus and minus), ++var, --var
- (type) Casting
- ! (Not)
- *, /, %
- +, - (Binary add and sub)
- <, <=, >, >= (Comparison)
- ==, !=; (Equality)
- && (Conditional AND)
- || (Conditional OR)
- +=, -=, *=, /=, %=  

Examples:
- x + a < y + b
- y <=a && z <=b

Lowest order
Example

Applying the operator precedence, the expression $3 + 4 \times 4 > 5 \times (4 + 3) - 1$ is evaluated as follows:

\[
\begin{align*}
3 + 4 & \times 4 > 5 \times (4 + 3) - 1 \\
3 + 4 & \times 4 > 5 \times 7 - 1 \\
3 + 16 & > 7 - 1 \\
3 + 16 & > 35 - 1 \\
19 & > 34 \\
\text{false}
\end{align*}
\]

or write $(3 + 4 \times 4) > (5 \times (4 + 3) - 1)$

Example: A Simple Math Learning Tool

This example creates a program to let a first grader practice additions. The program randomly generates two single-digit integers number1 and number2 and displays a question such as “What is 7 + 9?” to the student, as shown below. After the student types the answer in the input dialog box, the program displays a message dialog box to indicate whether the answer is true or false.
import javax.swing.*;

public class AdditionTutor {
    public static void main(String[] args) {
        int number1 = (int)(System.currentTimeMillis() % 10); //Fig 2.6
        int number2 = (int)(System.currentTimeMillis() * 7 % 10);

        String answerString = JOptionPane.showInputDialog
            ("What is " + number1 + " + " + number2 + "?");

        int answer = Integer.parseInt(answerString);

        JOptionPane.showMessageDialog(null,
            number1 + " + " + number2 + " = " + answer + " is " +
            (number1 + number2 == answer));
    }
}

**Selection Statements**

- *if Statements*
- *if ... else Statements*
- *switch Statements*
- *Conditional Operators*
Simple if Statements and the Flow Chart

if (booleanExpression) {
    statement(s);
}

if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area" + " for the circle of radius " + radius + " is " + area);
}

Note

Outer parentheses required

if ((i > 0) && (i < 10)) {
    System.out.println("i is an integer between 0 and 10");
}

(a)

Equivalent

Braces can be omitted if the block contains a single statement

if ((i > 0) && (i < 10))
    System.out.println("i is an integer between 0 and 10");

(b)
**Caution**

Adding a semicolon at the end of an if clause is a common mistake.

```java
if (radius >= 0);  
{
    area = radius*radius*PI;
    System.out.println("The area for the circle of radius "+
                        radius + "+" + area);
}
```

Only add a semicolon at the end of a statement, not after a condition!

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

This error often occurs when you use the next-line block style.

---

**The if...else Statement and the Flow Chart**

```java
if (booleanExpression) {
    statement(s)-for-the-true-case;
}
else {
    statement(s)-for-the-false-case;
}
```

![Flow Chart](image)

- **Boolean Expression**
- **true**
- **false**
- Statement(s) for the true case
- Statement(s) for the false case
**if...else Example**

```java
if (radius >= 0) {
    area = radius * radius * 3.14159;
    System.out.println("The area for the circle of radius "+ "is " + area);
} else {
    System.out.println("Negative input");
}
```

**Multiple Alternative if Statements (nested)**

```java
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

Suppose score is 70.0

Equivalent

```java
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```
### Note

The **else** clause matches the *most recent if* clause in the same block. Use braces for blocks wisely.

```java
int i = 1;
int j = 2;
int k = 3;
if (i > j)
    if (i > k)
        System.out.println("A");
    else
        System.out.println("B");
```

### TIPs and Caution

**if (number % 2 == 0)**

```java
if (number % 2 == 0)
    even = true;
else
    even = false;
```

**(boolean even = number % 2 == 0;**

```java
int i = 1;
int j = 2;
int k = 3;
if (i > j) {
    if (i > k)
        System.out.println("A");
} else
    System.out.println("B");
```

**What may happen?**

```java
if (even)
    System.out.println("It is even.");
```

**Equivalent**

```java
if (even == true)
    System.out.println("It is even.");
```

**Equivalent**

```java
if (even)
    System.out.println("It is even.");
```

**What may happen?**
Exercise

1. Suppose $x = 3$ and $y = 2$, show the output, if any, of the following code.

2. Suppose $x = 3$ and $y = 4$, show the output, if any, of the following code.

3. Draw a flowchart of the following code:

```java
if (x > 2) {
    if (y > 2) {
        int z = x + y;
        System.out.println("z is " + z);
    }
} else
    System.out.println("x is " + x);
```

Example: Computing Taxes

The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2002 are shown in Table 3.1.

<table>
<thead>
<tr>
<th>Tax rate</th>
<th>Single filers</th>
<th>Married filing jointly or qualifying widow/widower</th>
<th>Married filing separately</th>
<th>Head of household</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>Up to $6,000</td>
<td>Up to $12,000</td>
<td>Up to $5,000</td>
<td>Up to $10,000</td>
</tr>
<tr>
<td>15%</td>
<td>$6,001 - $27,950</td>
<td>$12,001 - $46,700</td>
<td>$5,001 - $23,350</td>
<td>$10,001 - $37,450</td>
</tr>
<tr>
<td>30%</td>
<td>$57,701 - $141,250</td>
<td>$112,951 - $171,950</td>
<td>$56,426 - $85,975</td>
<td>$96,701 - $156,600</td>
</tr>
<tr>
<td>35%</td>
<td>$141,251 - $307,050</td>
<td>$171,951 - $307,050</td>
<td>$85,976 - $153,525</td>
<td>$156,601 - $307,050</td>
</tr>
<tr>
<td>38.6%</td>
<td>$307,051 or more</td>
<td>$307,051 or more</td>
<td>$153,526 or more</td>
<td>$307,051 or more</td>
</tr>
</tbody>
</table>
Example: Computing Taxes, cont.

```
......
double tax = 0;
if (status == 0) {
    // Compute tax for single filers
    if (income <= 6000)
        tax = income * 0.10;
    else if (income <= 27950)
        tax = 6000 * 0.10 + (income - 6000) * 0.15;
    else if (income <= 67700)
        tax = ....
} else if (status == 1) {
    // Compute tax for married file jointly
} else if (status == 2) {
    // Compute tax for married file separately
} else if (status == 3) {
    // Compute tax for head of household
} else {
    // Display wrong status
}
```

Example: An Improved Math Learning Tool

Creates a program to teach a first grade child how to learn subtractions. The program randomly generates two single-digit integers number1 and number2 with number1 > number2 and displays a question such as “What is 9 – 2?” to the student, as shown in the figure.

For first grade child, negative number is not expected.

After the student types the answer in the input dialog box, the program displays a message dialog box to indicate whether the answer is correct, as shown in figure.
Code for the Improved Math Learning Tool

```java
import javax.swing.*; // or import javax.JOptionPane;

public class SubtractionTutor {
    public static void main(String[] args) {
        int number1 = (int)(System.currentTimeMillis() % 10);
        int number2 = (int)(System.currentTimeMillis() * 7 % 10);

        // if number1 < number2, switch number1 with number2
        if (number1 < number2) {
            ...
        }
        String answerString = JOptionPane.showInputDialog("What is \( + \) number1 + \( - \) number2 ?");
        int answer = Integer.parseInt(answerString);
        String replyString;
        if ...
        else ...
        JOptionPane.showMessageDialog(null, replyString);
    }
}
```

Re: Computing Taxes by if Statements

```java
double tax = 0;
if (status == 0) {
    // Compute tax for single filers
    if (income <= 6000)
        tax = income * 0.10;
    else if (income <= 27950)
        tax = 6000 * 0.10 + (income - 6000) * 0.15;
    else if (income <= 67700)
        tax = ....
}
else if (status == 1) {
    // Compute tax for married file jointly
}
else if (status == 2) {
    // Compute tax for married file separately
}
else if (status == 3) {
    // Compute tax for head of household
}
else {
    // Display wrong status
}
```

Overuse of if statement makes a program difficult to read.

Any better way to handle multiple conditions efficiently?
**switch Statements**

```java
switch (status) {
    case 0:  block of statements; //compute taxes for single filers
             break;
    case 1: block of statements; //compute taxes for married file jointly;
             break;
    case 2: block of statements; // compute taxes for married file separately
             break;
    case 3: block of statements; //compute taxes for head of household
             break;
    default: block of statements; // invalid status
}
```

**switch Statement Flow Chart**

What if the status of a case is 1? or 4?

- status is 0  
  - Compute tax for single filers  
  - break

- status is 1  
  - Compute tax for married file jointly  
  - break

- status is 2  
  - Compute tax for married file separately  
  - break

- status is 3  
  - Compute tax for head of household  
  - break

- default  
  - Default actions  
  - Next Statement
**switch** Statement Rules

The switch-expression must yield a value of **char**, **byte**, **short**, or **int** type and must always be enclosed in parentheses.

```java
switch (switch-expression) {
    case value1: statement(s)1;
    break;
    case value2: statement(s)2;
    break;
    ...
    case valueN: statement(s)N;
    break;
    default: statement(s)-for-default;
}
```

The `value1`, ..., and `valueN` must have the same data type as the value of the switch-expression.

The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression.

Note that `value1`, ..., and `valueN` are constant expressions, meaning that they cannot contain variables in the expression, such as $1 + x$.

**switch** Statement Rules (cont.)

The keyword **break** is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. If the break statement is not present, the next case statement will be executed.

```java
switch (switch-expression) {
    case value1: statement(s)1;
    break;
    case value2: statement(s)2;
    break;
    ...
    case valueN: statement(s)N;
    break;
    default: statement(s)-for-default;
}
```

The **default** case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.

The case statements are executed in sequential order, but the order of the cases does not matter. However, it is good programming style to follow the logical sequence of the cases and place the default case at the end.
### dayOfWeek Example

```java
public String weekDay(int dayOfWeek) {
    String dayName;
    switch (dayOfWeek) {
        case 2:  dayName = "Monday";        break;  // ensure immediate exit
        case 3:  dayName = "Tuesday";        break;
        case 4:  dayName = "Wednesday";      break;
        case 5:  dayName = "Thursday";        break;
        case 6:  dayName = "Friday";          break;
        default: dayName = "Unknown";         break;  // optional
    }
    return dayName;
}
```

Suppose `dayOfWeek` is 4:

### Exercise

What is the value of `y` after the following `switch` statement is executed?

```java
x = 3;
y = 3;
switch (x + 3) {
    case 6:  y = 1;
    default: y += 1;
}
System.out.println("y is " + y);
```
Conditional Operator / Shortcut

(booleanExpression) ? expression1 : expression2

Example1:

if (x > 0)
    y = 1;
else
    y = -1;

is equivalent to

Example2:

if (num % 2 == 0)
    System.out.println(num + "is even");
else
    System.out.println(num + "is odd");

Equivalent to?

System.out.println((num % 2 == 0)? num + "is even" : num + "is odd");

Formatting Output

Use the new JDK 1.5 printf statement; syntax:

    System.out.printf(format, item1, item2, ..... , itemk);

Where format is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

    String s = String.format("count is %d and amount is %f", x, y));
Frequently-Used Specifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Output</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>%b</td>
<td>a boolean value</td>
<td>true or false</td>
</tr>
<tr>
<td>%c</td>
<td>a character</td>
<td>'a'</td>
</tr>
<tr>
<td>%d</td>
<td>a decimal integer</td>
<td>200</td>
</tr>
<tr>
<td>%f</td>
<td>a floating-point number</td>
<td>45.460000</td>
</tr>
<tr>
<td>%e</td>
<td>a number in scientific notation</td>
<td>4.556000e+01</td>
</tr>
<tr>
<td>%s</td>
<td>a string</td>
<td>&quot;Java is cool&quot;</td>
</tr>
</tbody>
</table>

```java
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

display            count is 5 and amount is 45.560000
items

What we need to do if using System.out.println method?
How to keep a number of digits after the decimal point?

---

**Reading /Homework**

- Review the chapter 3 of the textbook
- Preview the chapter 4 of the textbook
- Do review questions: 3.2, 3.4, 3.10, 3.12, 3.15, 3.18, 3.19
- Do program examples in chapter 3