Type Conversion and Compatibility

- If a type is **compatible** to another, then the compiler will perform the conversion **implicitly**.
  - For example, the integer 1 is compatible to a double value 1.0.
- However, there is no automatic conversion from **double** to **int**. (Why?)
- To do so, you must use a **cast**, which performs an **explicit** conversion for compilation.
- Similarly, a **long** value is not compatible to **int**.
Casting

```java
...  
  int x = 1;
  double y = x; // compatible; implicit conversion
  x = y; // incompatible; need an explicit conversion by casting
  x = (int) y; // succeed!!
...```

- Note that the Java compiler does only type-checking but no real execution before compilation.
- In other words, the values of `x` and `y` are unknown until they are really executed.
Type Conversion and Compatibility (concluded)

- small-size types $\rightarrow$ large-size types
- small-size types $\leftrightarrow$ large-size types (need a cast)
- simple types $\rightarrow$ complicated types
- simple types $\leftrightarrow$ complicated types (need a cast)
A character stored by the machine is represented by a sequence of 0’s and 1’s.

- For example, ASCII code. (See the next page.)

- The `char` type is a 16-bit unsigned primitive data type.¹

¹Java uses **Unicode** to represent characters. Unicode defines a fully international character set that can represent all of the characters found in all human languages.
## ASCII (7-bit version)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec Char</th>
<th>Hex</th>
<th>Dec Char</th>
<th>Hex</th>
<th>Dec Char</th>
<th>Hex</th>
<th>Dec Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0</td>
<td>0x20</td>
<td>32</td>
<td>0x40</td>
<td>64</td>
<td>0x60</td>
<td>96</td>
</tr>
<tr>
<td>0x01</td>
<td>SOH</td>
<td>0x21</td>
<td>33</td>
<td>0x41</td>
<td>65</td>
<td>0x61</td>
<td>97</td>
</tr>
<tr>
<td>0x02</td>
<td>STX</td>
<td>0x22</td>
<td>34</td>
<td>0x42</td>
<td>66</td>
<td>0x62</td>
<td>98</td>
</tr>
<tr>
<td>0x03</td>
<td>ETX</td>
<td>0x23</td>
<td>35</td>
<td>0x43</td>
<td>67</td>
<td>0x63</td>
<td>99</td>
</tr>
<tr>
<td>0x04</td>
<td>EOT</td>
<td>0x24</td>
<td>36</td>
<td>0x44</td>
<td>68</td>
<td>0x64</td>
<td>100</td>
</tr>
<tr>
<td>0x05</td>
<td>ENQ</td>
<td>0x25</td>
<td>37</td>
<td>0x45</td>
<td>69</td>
<td>0x65</td>
<td>101</td>
</tr>
<tr>
<td>0x06</td>
<td>ACK</td>
<td>0x26</td>
<td>38</td>
<td>0x46</td>
<td>70</td>
<td>0x66</td>
<td>102</td>
</tr>
<tr>
<td>0x07</td>
<td>BEL</td>
<td>0x27</td>
<td>39</td>
<td>0x47</td>
<td>71</td>
<td>0x67</td>
<td>103</td>
</tr>
<tr>
<td>0x08</td>
<td>BS</td>
<td>0x28</td>
<td>40</td>
<td>0x48</td>
<td>72</td>
<td>0x68</td>
<td>104</td>
</tr>
<tr>
<td>0x09</td>
<td>HT</td>
<td>0x29</td>
<td>41</td>
<td>0x49</td>
<td>73</td>
<td>0x69</td>
<td>105</td>
</tr>
<tr>
<td>0x0A</td>
<td>LF</td>
<td>0x2A</td>
<td>42</td>
<td>0x4A</td>
<td>74</td>
<td>0x6A</td>
<td>106</td>
</tr>
<tr>
<td>0x0B</td>
<td>VT</td>
<td>0x2B</td>
<td>43</td>
<td>0x4B</td>
<td>75</td>
<td>0x6B</td>
<td>107</td>
</tr>
<tr>
<td>0x0C</td>
<td>FF</td>
<td>0x2C</td>
<td>44</td>
<td>0x4C</td>
<td>76</td>
<td>0x6C</td>
<td>108</td>
</tr>
<tr>
<td>0x0D</td>
<td>CAR</td>
<td>0x2D</td>
<td>45</td>
<td>0x4D</td>
<td>77</td>
<td>0x6D</td>
<td>109</td>
</tr>
<tr>
<td>0x0E</td>
<td>SO</td>
<td>0x2E</td>
<td>46</td>
<td>0x4E</td>
<td>78</td>
<td>0x6E</td>
<td>110</td>
</tr>
<tr>
<td>0x0F</td>
<td>SI</td>
<td>0x2F</td>
<td>47</td>
<td>0x4F</td>
<td>79</td>
<td>0x6F</td>
<td>111</td>
</tr>
<tr>
<td>0x10</td>
<td>DLE</td>
<td>0x30</td>
<td>48</td>
<td>0x50</td>
<td>80</td>
<td>0x70</td>
<td>112</td>
</tr>
<tr>
<td>0x11</td>
<td>DC1</td>
<td>0x31</td>
<td>49</td>
<td>0x51</td>
<td>81</td>
<td>0x71</td>
<td>113</td>
</tr>
<tr>
<td>0x12</td>
<td>DC2</td>
<td>0x32</td>
<td>50</td>
<td>0x52</td>
<td>82</td>
<td>0x72</td>
<td>114</td>
</tr>
<tr>
<td>0x13</td>
<td>DC3</td>
<td>0x33</td>
<td>51</td>
<td>0x53</td>
<td>83</td>
<td>0x73</td>
<td>115</td>
</tr>
<tr>
<td>0x14</td>
<td>DC4</td>
<td>0x34</td>
<td>52</td>
<td>0x54</td>
<td>84</td>
<td>0x74</td>
<td>116</td>
</tr>
<tr>
<td>0x15</td>
<td>NAK</td>
<td>0x35</td>
<td>53</td>
<td>0x55</td>
<td>85</td>
<td>0x75</td>
<td>117</td>
</tr>
<tr>
<td>0x16</td>
<td>SYN</td>
<td>0x36</td>
<td>54</td>
<td>0x56</td>
<td>86</td>
<td>0x76</td>
<td>118</td>
</tr>
<tr>
<td>0x17</td>
<td>ETB</td>
<td>0x37</td>
<td>55</td>
<td>0x57</td>
<td>87</td>
<td>0x77</td>
<td>119</td>
</tr>
<tr>
<td>0x18</td>
<td>CAN</td>
<td>0x38</td>
<td>56</td>
<td>0x58</td>
<td>88</td>
<td>0x78</td>
<td>120</td>
</tr>
<tr>
<td>0x19</td>
<td>EM</td>
<td>0x39</td>
<td>57</td>
<td>0x59</td>
<td>89</td>
<td>0x79</td>
<td>121</td>
</tr>
<tr>
<td>0x1A</td>
<td>SUB</td>
<td>0x3A</td>
<td>58</td>
<td>0x5A</td>
<td>90</td>
<td>0x7A</td>
<td>122</td>
</tr>
<tr>
<td>0x1B</td>
<td>ESC</td>
<td>0x3B</td>
<td>59</td>
<td>0x5B</td>
<td>91</td>
<td>0x7B</td>
<td>123</td>
</tr>
<tr>
<td>0x1C</td>
<td>FS</td>
<td>0x3C</td>
<td>60</td>
<td>0x5C</td>
<td>92</td>
<td>0x7C</td>
<td>124</td>
</tr>
<tr>
<td>0x1D</td>
<td>GS</td>
<td>0x3D</td>
<td>61</td>
<td>0x5D</td>
<td>93</td>
<td>0x7D</td>
<td>125</td>
</tr>
<tr>
<td>0x1E</td>
<td>RS</td>
<td>0x3E</td>
<td>62</td>
<td>0x5E</td>
<td>94</td>
<td>0x7E</td>
<td>126</td>
</tr>
<tr>
<td>0x1F</td>
<td>US</td>
<td>0x3F</td>
<td>63</td>
<td>0x5F</td>
<td>95</td>
<td>0x7F</td>
<td>127 DEL</td>
</tr>
</tbody>
</table>
Example

- Characters can also be used as an integer type on which you can perform arithmetic operations.²

- For example,

```java
... // A single-quoted value is the char type.
char x = 'a';
System.out.println(x + 1); // output 98!!
System.out.println((char)(x + 1)); // output 'b'
...
```

- Notice that a double-quoted string is a String object, which can have more char values.

Boolean Values

• The program is supposed to do decision making by itself, for example, Google Driverless Car.³
• To do this, Java has the boolean-type flow controls (selections and iterations).
• Only two possible values, true and false.
• Note that a boolean value cannot be cast into a value of another type, nor can a value of another type be cast into a boolean value.

³See https://www.google.com/selfdrivingcar/
Rational Operators

- These operators take two operands.
- Rational expressions return a boolean value.
- Note that the equality comparison operator is double equality sign (==), not single equality sign (=).

---

4See Table 3-1 in YDL, p. 82.
Be aware that e is logically correct but **syntactically wrong**.

Usually, the boolean expression consists of a **combination** of rational expressions.

- For example, $1 < x < 3$ should be $(1 < x)\&\&(x < 3)$, where $\&\&$ refers to the AND operator.
## Logical Operators

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

See Table 3-2 in YDL, p. 102.
• Let X and Y be two Boolean variables.
• Then the truth table for logical operators is as follows:

| X | Y | \!X | X&&Y | X || Y | X \land Y |
|---|---|-----|------|------|----------|
| T | T | F   | T    | T    | F        |
| T | F | F   | F    | T    | T        |
| F | T | T   | F    | T    | T        |
| F | F | T   | F    | F    | F        |

• Note that the instructions of computers, such as arithmetic operations, are implemented by logic gates.\(^6\)

\(^6\)See any textbook for digital circuit design.
“Logic is the anatomy of thought.”

– John Locke (1632–1704)

“This sentence is false.”

– anonymous

“I know that I know nothing.”

– Plato

(In Apology, Plato relates that Socrates accounts for his seeming wiser than any other person because he does not imagine that he knows what he does not know.)
### Arithmetic Compound Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>Increment</td>
</tr>
<tr>
<td>+=</td>
<td>Addition assignment</td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction assignment</td>
</tr>
<tr>
<td>*=</td>
<td>Multiplication assignment</td>
</tr>
<tr>
<td>/=</td>
<td>Division assignment</td>
</tr>
<tr>
<td>%=</td>
<td>Modulus assignment</td>
</tr>
<tr>
<td>-=</td>
<td>Decrement</td>
</tr>
</tbody>
</table>

- Note that these shorthand operators are not available in languages such as Matlab, R, and Python.
int x = 1;
System.out.println(x); // output 1
x = x + 1;
System.out.println(x); // output 2
x += 2;
System.out.println(x); // output 4
x++; // equivalent to x += 1 and x = x + 1
System.out.println(x); // output 5
• The compound assignment operators are also useful for char values.\textsuperscript{7}

• For example,

```java
... char s = 'a';
System.out.println(s); // output a
s += 1;
System.out.println(s); // output b
s++;System.out.println(s); // output c
...
```

\textsuperscript{7}Contribution by Mr. Edward Wang (Java265) on May 1, 2016.
++x vs. x++

• The expression ++x first increments the value of x and then returns x.

• Instead, the expression x++ first returns the value of x and then increments itself.

• For example,

```java
... int x = 1;
int y = ++x;
System.out.println(y); // output 2; aka preincrement
System.out.println(x); // output 2

int w = 1;
int z = w++;
System.out.println(z); // output 1; aka postincrement
System.out.println(w); // output 2
...
```

• We will use these notations very often.
Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>var++ and var-- (Postfix)</td>
</tr>
<tr>
<td></td>
<td>+, -- (Unary plus and minus), ++var and --var (Prefix)</td>
</tr>
<tr>
<td></td>
<td>(type) (Casting)</td>
</tr>
<tr>
<td></td>
<td>! (Not)</td>
</tr>
<tr>
<td></td>
<td>*, /, % (Multiplication, division, and remainder)</td>
</tr>
<tr>
<td></td>
<td>+, -- (Binary addition and subtraction)</td>
</tr>
<tr>
<td></td>
<td>&lt;, &lt;=, &gt;, &gt;= (Comparison)</td>
</tr>
<tr>
<td></td>
<td>==, != (Equality)</td>
</tr>
<tr>
<td></td>
<td>^ (Exclusive OR)</td>
</tr>
<tr>
<td></td>
<td>&amp;&amp; (AND)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>=, +=, -=, *=, /=, %= (Assignment operator)</td>
</tr>
</tbody>
</table>

See Table 3-10 in YDL, p. 116.
Using Parentheses

- Parentheses are used in expressions to change the natural order of precedence among the operators.
- One always evaluates the expression inside of parentheses first.
Scanner Objects

• It is not convenient to modify the source code and recompile it for a different radius.
• Reading from the console enables the program to receive an input from the user.
• A Scanner object provides some input methods, say the input received from the keyboard or the files.
• Java uses System.in to refer to the standard input device, by default, the keyboard.
Write a program which receives a number as input, and outputs the area of the circle.

```java
import java.util.Scanner;
...
Scanner input = new Scanner(System.in);
System.out.println("Enter r?");
// input
int r = input.nextInt();
// algorithm
double area = r * r * 3.14;
// output
System.out.println(area);
input.close();
...
In the listing, Line 3 is to create a **Scanner** object by the `new` operator, as an agent between the keyboard and your program.

Note that all objects are resided in the **heap** of the memory.

To control this object, its **memory address** is then assigned to the variable `input` which is a variable in the **stack** of memory.

So the variable `input` is a **reference**.

We will discuss the objects and reference variables later.
Methods Provided by Scanner Objects

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nextByte()</code></td>
<td>reads an integer of the <code>byte</code> type.</td>
</tr>
<tr>
<td><code>nextShort()</code></td>
<td>reads an integer of the <code>short</code> type.</td>
</tr>
<tr>
<td><code>nextInt()</code></td>
<td>reads an integer of the <code>int</code> type.</td>
</tr>
<tr>
<td><code>nextLong()</code></td>
<td>reads an integer of the <code>long</code> type.</td>
</tr>
<tr>
<td><code>nextFloat()</code></td>
<td>reads a number of the <code>float</code> type.</td>
</tr>
<tr>
<td><code>nextDouble()</code></td>
<td>reads a number of the <code>double</code> type.</td>
</tr>
<tr>
<td><code>next()</code></td>
<td>reads a string that ends before a whitespace character.</td>
</tr>
<tr>
<td><code>.nextLine()</code></td>
<td>reads a line of text (i.e., a string ending with the <code>Enter</code> key pressed).</td>
</tr>
</tbody>
</table>

---

\(^9\)See Table 2-1 in YDL, p. 38.
Example: Mean and Standard Deviation

Write a program which calculates the mean and the standard deviation of 3 numbers.

• The mean of 3 numbers is given by $\bar{x} = \left( \sum_{i=1}^{3} x_i \right) / 3$.
• Also, the resulting standard deviation is given by

$$S = \sqrt{\frac{\sum_{i=1}^{3} (x_i - \bar{x})^2}{3}}.$$

• You may use these two methods:
  • Math.pow(double x, double y) for $x^y$
  • Math.sqrt(double x) for $\sqrt{x}$

• See more methods within Math class.
Scanner input = new Scanner(System.in);
System.out.println("a = ?");
double a = input.nextDouble();
System.out.println("b = ?");
double b = input.nextDouble();
System.out.println("c = ?");
double c = input.nextDouble();

double mean = (a + b + c) / 3;
double std = Math.sqrt((Math.pow(a - mean, 2) +
    Math.pow(b - mean, 2) +
    Math.pow(c - mean, 2)) / 3);

System.out.println("mean = " + mean);
System.out.println("std = " + std);
class Lecture3 {

    "Selections"

}

// Keywords
if, else, else if, switch, case, default
Flow Controls

The basic algorithm (and program) is constituted by the following operations:

- **Sequential statements**: execute instructions in order.
- **Selection**: first check if the predetermined condition is satisfied, then execute the corresponding instruction.
- **Repetition**: repeat the execution of some instructions until the criterion fails.
• Note that they are involved with each other generally.
• For example, recall how to find the maximum in the input list?
Selections

- One-way `if` statements
- Two-way `if-else` statements
- Nested `if` statements
- Multiway `if-else if-else` statements
- `switch-case` statements
- Conditional operators
A one-way if statement executes an action if and only if the condition is true.
• The keyword if is followed by the parenthesized condition.
• The condition should be a boolean expression or a boolean value.
• It the condition is true, then the statements in the selection body will be executed once.
• If not, then the program won’t enter the selection body and skip the whole selection body.
• Note that the braces can be omitted if the block contains only single statement.
Example

Write a program which receives a nonnegative number as input for the radius of a circle, and determines the area of the circle.

```java
... double area;
if (r > 0) {
    area = r * r * 3.14;
    System.out.println(area);
}
... 
```

- However, the world is not well-defined.
A two-way if-else statement decides which statements to execute based on whether the condition is true or false.

```java
...  
if (condition) {
    // body for the true case
} else {
    // body for the false case
}  
...  
```
A flowchart illustrating decision-making in programming:

- The boolean expression is evaluated.
- If true, fall through to the statement(s) for the true case.
- If false, fall through to the statement(s) for the false case.

The flowchart shows a diamond-shaped decision block with 'true' and 'false' as outcomes, leading to the respective statement blocks.
Write a program which receives a number as input for the radius of a circle. If the number is nonnegative, then determine the area of the circle; otherwise, output “Not a circle.”

```java
... double area;
if (r > 0) {
    area = r * r * 3.14;
    System.out.println(area);
} else {
    System.out.println("Not a circle.");
}
input.close();
...
Nested if Statements

- For example,

```java
... 
    if (score >= 90)
        System.out.println("A");
    else {
        if (score >= 80)
            System.out.println("B");
        else {
            if (score >= 70)
                System.out.println("C");
            else {
                if (score >= 60)
                    System.out.println("D");
                else
                    System.out.println("F");
            }
        }
    }
... 
```
Multi-Way if-else

- Let’s redo the previous problem.

```java
... 
if (score >= 90) 
    System.out.println("A");
else if (score >= 80) 
    System.out.println("B");
else if (score >= 70) 
    System.out.println("C");
else if (score >= 60) 
    System.out.println("D");
else 
    System.out.println("F");
...
```

- An if-elseif-else statement is a preferred format for multiple alternatives, in order to avoid deep indentation and make the program easy to read.
• The order of conditions may be relevant. (Why?)

```java
... if ((score >= 90) && (score <= 100))
else if ((score >= 80) && (score < 90))
... else
...```

• The performance may degrade due to the order of conditions. (Why?)
Common Errors

```java
...  
  double area;
  if (r > 0);
      area = r * r * 3.14;
  System.out.println(area);
...  
```
Example

Generating random numbers

Write a program which generates 2 random integers and asks the user to answer the math expression.

- For example, the program shows $2 + 5 = ?$
- If the user answers 7, then the program reports “Correct.” and terminates.
- Otherwise, the program reports “Wrong answer. The correct answer is 7.” for this case.
- You may use `Math.random()` for a random value between 0.0 and 1.0, excluding themselves.
```
...  
int x = (int) (Math.random() * 10); // integers 0 ~ 9
int y = (int) (Math.random() * 10);
int answer = x + y;

System.out.println(x + " + " + y + " = ?");

Scanner input = new Scanner(System.in);
int z = input.nextInt();

if (z == answer)
    System.out.println("Correct.");
else
    System.out.println("Wrong. Answer: " + answer);
input.close();
...
```

- Can you extend this program for all arithmetic expressions (i.e., + − × ÷)?
Exercise

Find Max

Write a program which determines the maximum value in 3 random integers whose range from 0 to 99.

• How many variables do we need?
• How to compare?
• How to keep the maximum value?
• In this case, a scalar variable is not convenient. (Why?)
• So we need arrays and loops.
A switch-case structure takes actions depending on the target variable.

```java
... switch (target) {
    case v1:
        // statements
        break;
    case v2:
    case vk:
        // statements
        break;
    default:
        // statements
}
... 
```
• A switch-case statement is more convenient than an if statement for multiple discrete conditions.
• The variable target, always enclosed in parentheses, must yield a value of char, byte, short, int, or String type.
• The value $v_1, \ldots, v_k$ must have the same data type as the variable target.
• In each case, a break statement is a must.\(^{10}\)
  • break is used to break a construct!
• The default case, which is optional, can be used to perform actions when none of the specified cases matches target.
  • Counterpart to else statements.

\(^{10}\)If not, there will be a fall-through behavior.
...  

// RED: 0
// YELLOW: 1
// GREEN: 2

int trafficLight = (int) (Math.random() * 3);
switch (trafficLight) {
    case 0:
        System.out.println("Stop!!!");
        break;
    case 1:
        System.out.println("Slow down!!");
        break;
    case 2:
        System.out.println("Go!");
}
...
Conditional Operators

A conditional expression evaluates an expression based on the specified condition and returns a value accordingly.

```java
... someVar = booleanExpr ? exprA : exprB;
...```

- This is the only ternary operator in Java.
- If the `boolean` expression is evaluated `true`, then return `exprA`; otherwise, `exprB`. 

- For example,

```java
...
    if (num1 > num2)
        max = num1;
    else
        max = num2;
...
```

- Alternatively, one can use a conditional expression like this:

```java
...
    max = (num1 > num2) ? num1 : num2;
...
```
class Lecture4 {
    "Loops"
}

// keywords:
while, do, for, break, continue
Loops

A loop can be used to make a program execute statements repeatedly without having to code the same statements.

• For example, a program outputs “Hello, Java.” for 100 times.

```java
... System.out.println("Hello, Java.");
System.out.println("Hello, Java.");
...
  // copy and paste for 100 times
...
System.out.println("Hello, Java.");
...```

Zheng-Liang Lu  Java Programming 112 / 116
This is a simple example to show the power of loops.

In practice, any routine which repeats couples of times can be done by folding them into a loop.

\[\text{int cnt = 0;}\]
\[\text{while (cnt < 100) { } }\]
\[\quad \text{System.out.println("Hello, Java."); }\]
\[\quad \text{cnt++; }\]
\[\} \]

\[\text{...}\]
Loops provide substantial computational power.
Loops bring an efficient way of programming.
Loops could consume a lot of time.\textsuperscript{12}

\textsuperscript{12}We will visit the analysis of algorithms in the end of this lecture.
while Loops

A `while` loop executes statements repeatedly while the condition is true.

```
...  
while (condition) {
    // loop body
}
...  
```

- The condition should be a boolean expression which determines whether or not the execution of the body occurs.
- If true, the loop body is executed and check the condition again.
- Otherwise, the entire loop terminates.
count = 0;

(count < 100)?

ture

System.out.println("Welcome to Java!");
count++;

false