Review

Example

- When you to top up your mobile phone, you have to go to the counter to tell the teller what is your mobile phone, and amount of money you want to top up. The teller key the information in the computer then you pay the money. If you pay more then the amount you want, you will get a change

Example

- Write program to receive the number, and amount to pay, then the computer print the number and amount for the teller to ask for the confirmation from the customer. Then, the teller have to input the amount of money paid and show the change to tell the customer

Boolean Logic

Type boolean

- Type boolean has two values: true and false
- The associated operators:

'&&' -- 'and'

'||' -- 'or'

'!' -- 'not'

Type boolean

- 'true && true' has the value true
- 'false || true' has the value true
- '!true' has the value false

Type boolean

- x && y is true if both operands are true.
- x || y is false when both operands are false.
- ! (not) has highest precedence, followed by && and finally ||.

Type boolean

- The truth table

x	y	x && y (and)	x || y (or)	!x (not)
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Relational Operators

- Java provides a set of relational operators, used in expressions that evaluate to true or false.

- Each relational operator requires two operands
  - two integers
  - two decimal numbers
  - two characters.

Relational Operators

The relational operators are:

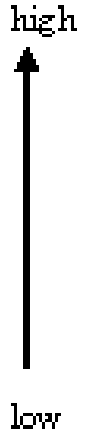
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equals (has the same value)
!=	not equal

Relational Operators

- Character data are compared based upon Unicode /ASCII integer values.
- ‘A’ has the code value 65 and ‘C’ the value 67
- ‘A’ < ‘C’ evaluates to true since 65 <67

Relational Operators

Operator	Associativity
!	**Right to left**
* / %	Left to right
+ -	Left to right
< <= > >=	Left to right
== !=	Left to right
&&	Left to right
||	Left to right



Relational Operators

Examples:

1. **5 < 3 || 6 > 2**
2. **1+14 % 5 == 0**
3. **'A' <'B'**
4. **'Z' < 'a'**
5. **1+1==2 ||1+1 == 3**

Relational Operators

Examples:

1. 5 < 3 || 6 > 2 false || true => true
2. 1+14 % 5 == 0 false
3. 'A' <'B' true (code for 'A' is 65; for B it is 66 : 65 < 66)
4. 'Z' < 'a' true (code for ‘Z’ is 90; and for 'a' it is 97:)
5. 1+1==2 ||1+1 == 3 true || false => true

6. 37/3 > .3333
7. 2<3 &&4<5 || 7<=5 && 2==3
8. 2<3 && (4<5 || 7<=5) && 2==3
9. false == false
10. true != false

6. 37/3 > .3333 true

7. 2<3 &&4<5 || 7<=5 && 2==3 true || false => true

8. 2<3 && (4<5 || 7<=5) && 2==3 true && true && false => false

9. false == false true

10. true != false true

Relational Operators

- Expressions such as 2<3<4 are incorrect.
- Java attempts to evaluate this expression as:

(2<3) < 4

true < 4

- 'true < 4' is invalid and generates an error

- The Java equivalent of 2<3<4 is (2<3) && (3<4)

Relational Operators

Problem Statement

A leap year is any year divisible by four except those years divisible by 100 unless the year is also divisible by 400."

Write a program that determines whether or not 1800 was a leap year.

Relational Operators

```
1.  // A leap year is a year that is divisible by 4
    // but not 100 unless it is divisible by 400
2.  // This program determines whether or not
    //1800 meets all conditions of a leap year
3.  public class LeapYear
4.  {
5.      public static void main(String[] args)
6.      {
7.          System.out.print("The year 1800 is a leap year? True or false: ");
8.          //(divisible by 4 and not by 100) or  (divisible by 400)
9.          System.out.println( 1800%4 ==0 && 1800 %100 !=0 ||
                                    1800%400 == 0);
10.     }
11. }
```

Relational Operators

- **Output**

The year 1800 is a leap year? True or false: false

Here is a fully parenthesized version of the expression on line 9:

(((1800%4) ==0) && ((1800 %100) !=0)) || ((1800%400) == 0)

Short Circuit Evaluation

- Evaluates expressions involving operators && and ||.

- Java stops the evaluation of an expression once the value of the expression is determined.
- Example:

  1<2 || 2<3 || 3<4
- The value of this expression is true
- This value can be determined after evaluating 1<2
- No more of the expression need be considered.

Mixing Data Types in a Numerical Expression

- A Java expression can be constructed from data of several different types
- The expression (22+3.0)/4 contains both integers (int) and decimal numbers (double)
- Java first promotes or *casts* the operand of the "smaller" data type to the data type of the other operand
- The range of values determines the "size" of a data type
  - char is smaller than int
  - int is smaller than double.

Mixing Data Types in a Numerical Expression

- Example:

  The expression (22+3.0)/4 has the value 6.25.

The expression is evaluated as follows:

(22+3.0)/4 The expression consists of decimal and integer types.

(22.0 +3.0)/4 The integer 22 is cast to 22.0 (an int is cast to a double).

25.0/4 This is floating point addition -- 22.0+3.0.

25.0/4.0 The integer 4 is cast to 4.0.

Mixing Data Types in a Numerical Expression

The expression 'A' + 1 has the value 66.

The expression is evaluated as follows:

- 'A' + 1 The expression consists of character and integer data.
- 65 + 1 The character 'A' is cast to the integer 65 -- its ASCII code value.
- 66 This is integer addition.

Mixing Data Types in a Numerical Expression

The expression 'A' + 1.0 has the value 66.0.

The expression is evaluated as follows:

- 'A' + 1.0 The expression consists of character and a decimal data.
- 65 + 1.0 The integer 65 is the ASCII code for 'A' i.e.
  - 'A' is stored as 65.
- 65.0 + 1.0 The integer 65 is cast to 65.0.
- 66.0 This is floating point addition.

The + Operator and Strings

- A+B
- if A and B are string
  - the expression A+B evaluates to another string
  - the concatenation (joining together) of A and B.

- If only one operand is a string
  - then the other operand is cast to a string
  - The value of the expression is the concatenation of two strings.

The + Operator and Strings

- Joining a string and a number:

The expression:

2147483647 + " is not only the largest value of type int but also a prime number!"

evaluates to the string:
"2147483647 is not only the largest value of type int but also a prime number!"

- The first operand is the integer 2147483647
  - cast to the string "2147483647"
  - the two strings are concatenated.

The + Operator and Strings

- Joining a string and a numerical expression:

The expression “The sum of the two dice is " + (5+2)

evaluates to the string:

“The sum of the two dice is 7”

- Parentheses can force a change in the usual precedence.

The + Operator and Strings

- If the parentheses are omitted then the expression:

  "The sum of the two dice is "+ 5 + 2

evaluates to the string:

  "**The sum of the two dice is 52.**"

The + Operator and Strings

("The sum of the two dice is " + 5)+ 2

("The sum of the two dice is " + "5") + 2	The integer 5 is cast to string "5".
("The sum of the two dice is 5") + 2	"The sum of the two dice is " and "5“ are joined.
("The sum of the two dice is 5")+ "2"	The integer 2 is cast to "2".
"The sum of the two dice is 52"	"The sum of the two dice is 5" is concatenated with "2".

Numerical addition is not performed.

The + Operator and Strings

The expression:

"The product of the two dice is " + 5*2

Evaluates to the string:

"The product of the two dice is 10."

The * operation is performed first since * has higher precedence than +.

The + Operator and Strings

The expression

"The difference of the two dice is" + 5 – 2

is ill formed and causes an error.

Left to right the expression is evaluated as:

("The difference of the two dice is " + 5)– 2

("The difference of the two dice is "+ "5") – 2

"The difference of the two dice is 5"– 2

Error occurs: the (–) operator cannot be applied to strings.

Bugs

- Initial versions of almost every program commonly contain errors or bugs.

- There are three categories of errors:

    1. Compilation errors.
    2. Runtime errors.
    3. Logical errors.

Compilation Errors

- When a program violates one of the rules of Java
  - The omission of a semi-colon
  - A string's quotation mark, or a closing curly brace.

- The compiler flags the error
  - Tell you where the error occurs.

- A program must be free of such errors before it can be translated into bytecode.

Runtime Errors

- A runtime error occurs during program execution.
- A runtime error can occur when the program attempts some invalid operation
  - division by zero.
- A runtime error results in program termination.

Logical Errors

- Even if the Java compiler detects no errors and a program runs to completion
  - a program may not do what it is supposed to do.
- A program that converts degrees Fahrenheit to degrees Celsius using the erroneous expression:

(5/9)(F – 32), where F represents a Fahrenheit temperature;

// evaluates to 0 since 5/9 = 0 (integer division)

rather than:

(5.0)/9.0)(F – 32) // correct, uses floating point division

Activities

- Evaluate the following expression
  - 2 < 3
  - 2 < 3 || 2-9 > 0
  - 2-9 > 0 == false