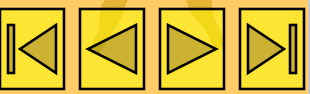
Introduction to Computing Using Java

Primitive Data Type Boolean and Related Operators



Boolean/Truth/Logic Processing

- 布林(布爾)/真值/邏輯運算
- A proposition may be true, false or undetermined, e.g.
  - true/false questions in examinations
  - You are a boy
  - Peter is 21 years old
  - Principal of no less than HK$30,000
  - It will rain tomorrow?

Boolean Values

```
int     moneyAtHand   = 3000;
boolean noMoneyAtHand = false;

accountMichael.deposit(3000);
moneyAtHand   = moneyAtHand - 3000;

noMoneyAtHand = true;
noMoneyAtHand = moneyAtHand <= 0;
```

- The only values a boolean type field can take is either true or false.

Expressions

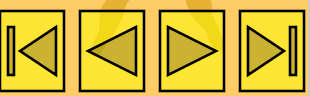
- Up to now we have seen
  - *arithmetic expressions* that use the operators

`+ - * / % ++ --`

  - *assignment expressions* that use the operators

`= += -=` …

- ***Boolean expressions*** use *relational* and *logical* operators.
- The **result** of a Boolean expression is either `true` or `false`.
- Boolean expressions allow us to write programs that decide whether to execute some code or not.
- These decisions changes the *flow* of the program execution.



Relational Operators

- ***Relational operators*** compare two **arithmetic expressions** and evaluate to a **`boolean`** result.

★==	LHS is equal to RHS
★!=	LHS is not equal to RHS
★>	LHS is greater than RHS
★<	LHS is less than RHS
★>=	LHS is greater than or equal to RHS
★<=	LHS is less than or equal to RHS
★=	Assignment/Storage Operation! LHS is a locker!

This is NOT a relational operator!

Relational Operators

- How do they look like?
  - For numerics:

```
3 > 7
-1 != 1
oldWeight + oldHeight >= 190.34 * newHeight
18 == Age
```

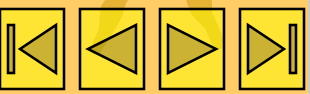
  - For characters:

```
'Z' > 'A'
'A' > '0'       ??? true ???
'a' > 'A'       ??? true ???
```

- Result must be true or false

Relational operators

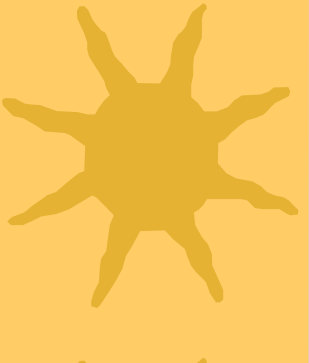
- These relational operators have **lower precedence** than the arithmetic operators.
  - Thus, arithmetic expressions are evaluated first, then the resulting Boolean expressions.
  - That is, Java does the “math” first, then the comparison.

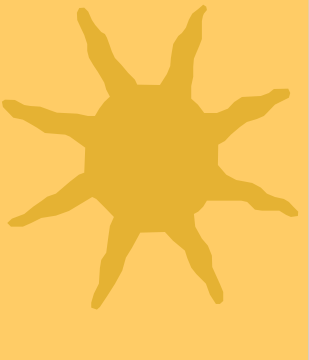


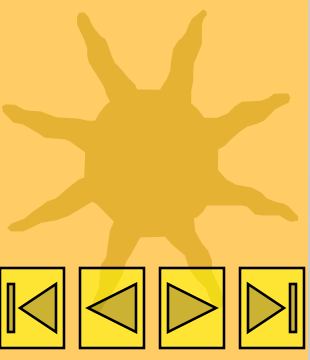
Relational operators

Examples:

```
int x = 15;
int y = 100;
   System.out.println(x > y);
   System.out.println(x < 15);
   System.out.println(x <= 15)
   System.out.println(x == y);
   System.out.println(x != 5);
   System.out.println(x * -y > 0);
   boolean isBigger = x > y;
```







Calculate firstly

Floating point comparison

- True or false? (inexact bit storage)
  - `0.7 * 0.7 == 0.49` `??? false ???`
- Why false?
  - Numerical error
- How to compare floating point?
  - **`Math.abs(f1-f2) < threshold`** **`(e.g. 0.0000001)`**
  - Threshold: a very small value

Logical operators

- ***Logical operators*** combine `boolean` values and evaluate to a `boolean` result.

Operator	Name	Example	Result
`!`	Logical NOT	`!a`	true if `a` is false, false if `a` is true
`&&`	Logical AND	`a && b`	true if both `a` and `b` are true, false otherwise
`||`	Logical OR	`a || b`	true if `a` or `b`, or both are true, false otherwise

Boolean Expression

```
boolean iAmDry;
/*
 * sunnyDay and noSweat → iAmDry
 */
iAmDry = sunnyDay && noSweat;
--- 8< ------------------------
```

What if it's NOT sunny? `[iAmDry=false]`

What if I DO sweat? `[iAmDry=false]`

Boolean AND (&)

`iAmDry`	`noSweat`		
	and (&)	true	false
`sunnyDay`	true	true	false
	false	false	false

Boolean OR (|)

phoneRing	callComing		
lowBattery	or (|)	true	false
	true	true	true
	false	true	false

Short Circuit (AND)

iAmDry	noSweat		
	and (&&)	true	false
	true	true	false
sunnyDay	false	false	false

Short Circuit (OR)

`phoneRing`	`callComing`		
	or (||)	true	false
`lowBattery`	true	true	true
	false	true	false

Short Circuit Boolean Operators (&& ||)

```
iAmDry = sunnyDay && noSweat
```

- If **sunnyDay** is false, Java will not check the truth value of **noSweat** as the result must be false.

```
phoneRing = lowBattery || callComing;
```

- If **lowBattery** is true, Java will not check the truth value of **callComing** as the result must be true.

- This is called ***short circuit*** boolean evaluation.

Other Boolean Operators

- The *not (!)* operator gives you the negation (反話).

```
iAmCareless = !iAmCareful;
```

- The *xor (^)* exclusive-or operator gives you false when the truth values of both operands are equal, true otherwise.

```
normalDay = workingDay ^ HOLIDAY;
     true <- false ^ true
     true <-  true ^ false
    false <- false ^ false
    false <-  true ^ true
```

Truth Tables

- ***Truth tables*** list all possible combination of values for the variables in an expression.

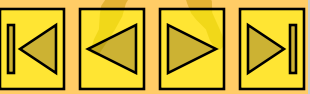
a	b	a && b	a || b	!a
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Logical operators

Example:

`age > 26`	`hasLicense`	`(age > 26) && hasLicense`

```
boolean canRentCar = (age > 26) && hasLicense;
```



Logical operators

Example:

age > 26	hasLicense	(age > 26) && hasLicense
true	true	true
true	false	false
false	**true**	**false**
false	false	false

```
int age = 16;
boolean hasLicense = true;
boolean canRentCar = (age > 26) && hasLicense;
```

Logical operators: Exercise 1

- It is time to buy a Iphone X when at least one of the following situations occurs:
  - the phone breaks
  - the phone is at least 3 years old

```
int phoneAge;          // in years
boolean isBroken;
…                      // code initializes variables

boolean needPhone = ____________________________;
```

Logical operators: Exercise 1

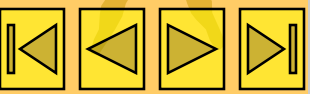
- It is time to buy a new phone when at least one of the following situations occurs:
  - the phone breaks
  - the phone is at least 3 years old

```
int phoneAge;          // in years
boolean isBroken;
…                      // code initializes variables

boolean needPhone = (isBroken == true)
                         || (phoneAge >= 3);
```

Logical Operators: Exercise 2

- Assume `x`, `y`, and `z` are `int` variables that have been initialized.

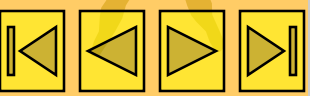
```
boolean areAllEqual =_____;
```



Logical Operators: Exercise 2

- Assume `x`, `y`, and `z` are `int` variables that have been initialized.

```
boolean areAllEqual = (x == y) && (y == z);
```



Logical operators

Examples:

`&&` is evaluated after relational operators.
`||` is evaluated after `&&`.

```
int x = 15;

int y = 100;

System.out.println(x > y && x >= 15);

System.out.println(x < 15 || x > 15);

System.out.println(x == y && y == 100);

System.out.println(x != 5 && x < y);

System.out.println(x + y > 100 || y <= 10);
```

Complex Boolean Expression

```
iAmDry       = sunnyDay && noSweat;
phoneSilent = lowBattery && callForwarded;
badMood      = !(iAmDry && phoneSilent);
/* by De Morgan's Law: & ←→ | */
badMood      = !iAmDry || !phoneSilent;
/* in expanded form: */
badMood = ! (sunnyDay && noSweat) |
          ! (lowBattery && callForwarded);
```

Boolean Algebra

- Double negative: `!!a` ≡ `a`
- de Morgan's Law:

```
!(a && b) ≡ !a || !b
!(a || b) ≡ !a && !b
```

de Morgan's Law (version 1)

Truth table: Consider all possible combinations of values of `boolean` **`a`** and **`b`**.

```
!(a && b) == (!a || !b)
```

a	b	a && b	!(a && b)	!a	!b	!a || !b
T	T					
T	F					
F	T					
F	F					

de Morgan's Law (version 1)

Truth table: Consider all possible combinations of values of `boolean` **`a`** and **`b`**.

```
!(a && b) == (!a || !b)
```

equal

a	b	a && b	!(a && b)	!a	!b	!a || !b
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

de Morgan's Law (version 2)

Truth table: Consider all possible combinations of values of `boolean` **`a`** and **`b`**.

```
!(a || b) == (!a && !b)
```

a	b	a || b	!(a || b)	!a	!b	!a && !b
T	T					
T	F					
F	T					
F	F					

de Morgan's Law (version 2)

Truth table: Consider all possible combinations of values of `boolean` **`a`** and **`b`**.

```
!(a || b) == (!a && !b)
```

equal

a	b	a || b	!(a || b)	!a	!b	!a && !b
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

de Morgan's Law

In Java:

```
!((age < 12) || (age >= 65))
```

In English: *It is not the case that age less than 12 or age greater than or equal to 65. !!!?*

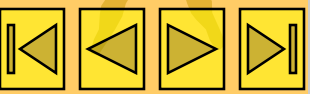
Simplify using de Morgan's Law:

```
!(age < 12) && !(age >= 65)
```

The reverse the meaning of the relational expressions:

```
(age >= 12) && (age < 65)
```

That is, *when age is at least 12 and less than 65.*



de Morgan's Law

In English:

Words neither rhyme nor alliterate.

In Java:

```
!wordsRhyme  &&  !wordsAlliterate
```

Words don't rhyme and they don't alliterate

Apply de Morgan's Law:

```
!(wordsRhyme || wordsAlliterate)
```

It's not the case words rhyme or alliterate.

Precedence

- ( ) [highest]
- !
- &
- |
- &&
- | | [lowest]

More About Expression

- Assignment is also an expression!

```
USdollar = (HKdollar = 123.45) * rate;
```

[double = double * double]

- Left to right evaluation

$1 - 2 - 3 \;=\; (1 - 2) - 3 \;\neq\; 1 - (2 - 3)$

- Short forms for number types

```
Age = Age + 1; ≡ Age++;
Age = Age - 1; ≡ Age--;
```

- Short forms for number types

```
Age = Age + 15;              ≡ Age += 15;
USdollar = USdollar - 1.7; ≡ USdollar -= 1.7;
```

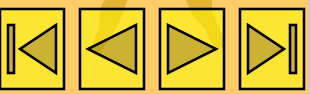
Mixing Numeric and Boolean Expressions

```
int temperature = 38;

boolean goHome = (temperature > 40) | (temperature < 0);
System.out.println("goHome? " + goHome);

int diceOne = 5, diceTwo = 6, diceThree = 6;

boolean triple = (diceOne == diceTwo) &
                 (diceTwo == diceThree);
boolean small  = (diceOne + diceTwo + diceThree <= 10) &
                 ! triple;
boolean big    = (diceOne + diceTwo + diceThree >= 11) &
                 ! triple;
```

Prime Number Checker

Get an integer input from user through console (remember how to use Scanner?), then check that whether this input number is a prime number or not?

```
boolean isPrime = true;
Scanner scan = new Scanner(System.in);
int num = scan.nextInt();
int temp = 0;
```



Prime Number Checker

```
for(int i = 2; i <= Math.floor(Math.sqrt(num)); i++)
{
        temp = num % i;
        if(temp == 0)
        {
                isPrime = false;
                break;
        }
}
```

Exercise: MPF Contribution

- http://www.mpfa.org.hk/eng/mpf_system/system_features/contributions/index.jsp
- Write a Java program to calculate the monthly mandatory contribution of an employee, given input the relevant monthly salary.

Introduction—MPF

- Mandatory Provident Fund (**強積金制度**)

Relevant month salary	Monthly mandatory contribution (employee)
Less than $7,100	No contributions required
Between $7,100 to $30,000	Relevant salary*5%
More than $30,000	$1,500