# Types, Variables and Operators

Computer Engineering Department
Java Course

Asst. Prof. Dr. Ahmet Sayar
Kocaeli University - Fall 2013

# Types

- Kinds of values that can be stored and manipulated
- **boolean: Truth value (true or false).**
- **int: Integer (0, 1, -47).**
- **double: Real number (3.14, 1.0, -2.1).**
- **String: Text (“hello”, “example”).**

# Variables

- Named location that stores a value of one particular type
- Form:
  - ***TYPE NAME;***
- Example:
  - String foo;
  - int x;
- A variable must be declared before it is used.

# Java Identifiers

- An *identifier* is a name, such as the name of a variable.
- Identifiers may contain only
  - letters
  - digits (0 through 9)
  - the underscore character (_)
  - and the dollar sign symbol ($) which has a special meaning

but the first character cannot be a digit.

# Example identifiers: Check their correctness

- int k!34;
- int 2dfg;
- int test1;
- int test23we;
- int df_;
- int sd$;
- int @kl;
- int $fg;
- int k.t;
- int k-t;

# Java Identifiers, cont.

- Identifiers may not contain any spaces, dots (`.`), asterisks (`*`), or other characters:

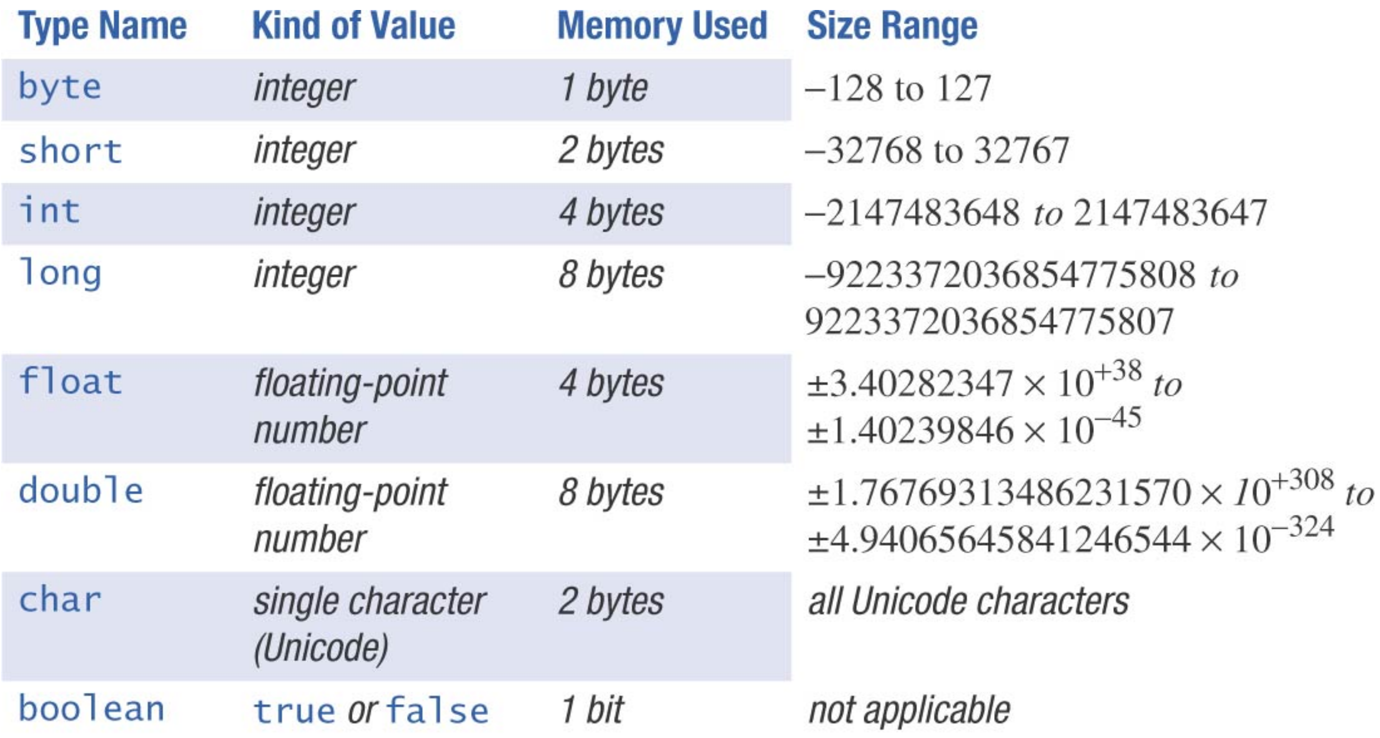
  `7-11  netscape.com  util.*` (not allowed)

- Identifiers can be arbitrarily long.
- Since Java is *case sensitive*, `stuff`, `Stuff`, and `STUFF` are different identifiers.

# Keywords or Reserved Words

- Words such as **if** are called *keywords* or *reserved words* and have special, predefined meanings.
- Keywords cannot be used as identifiers.
- Other keywords: `int, public, class`

# Primitive Types



| Type Name | Kind of Value | Memory Used | Size Range |
|---|---|---|---|
| `byte` | *integer* | *1 byte* | –128 to 127 |
| `short` | *integer* | *2 bytes* | –32768 to 32767 |
| `int` | *integer* | *4 bytes* | –2147483648 *to* 2147483647 |
| `long` | *integer* | *8 bytes* | –9223372036854775808 *to*<br>9223372036854775807 |
| `float` | *floating-point number* | *4 bytes* | $\pm 3.40282347 \times 10^{+38}$ *to*<br>$\pm 1.40239846 \times 10^{-45}$ |
| `double` | *floating-point number* | *8 bytes* | $\pm 1.76769313486231570 \times 10^{+308}$ *to*<br>$\pm 4.94065645841246544 \times 10^{-324}$ |
| `char` | *single character (Unicode)* | *2 bytes* | *all Unicode characters* |
| `boolean` | `true` *or* `false` | *1 bit* | *not applicable* |

Display 2.2

Primitive Types

# Assignment

- An assignment statement is used to assign a value to a variable.

  ```
  answer = 42;
  ```

- Use ‘**=**‘ to give variables a value.
- Example:
  - String foo;
  - foo = “IAP 6.092”;
- Can be combined with variable declaration
  - String foo = “IAP 6.092”;
- int numberOfBaskets, eggsPerBasket;
- int numberOfBaskets=5, eggsPerBasket;

# Operators

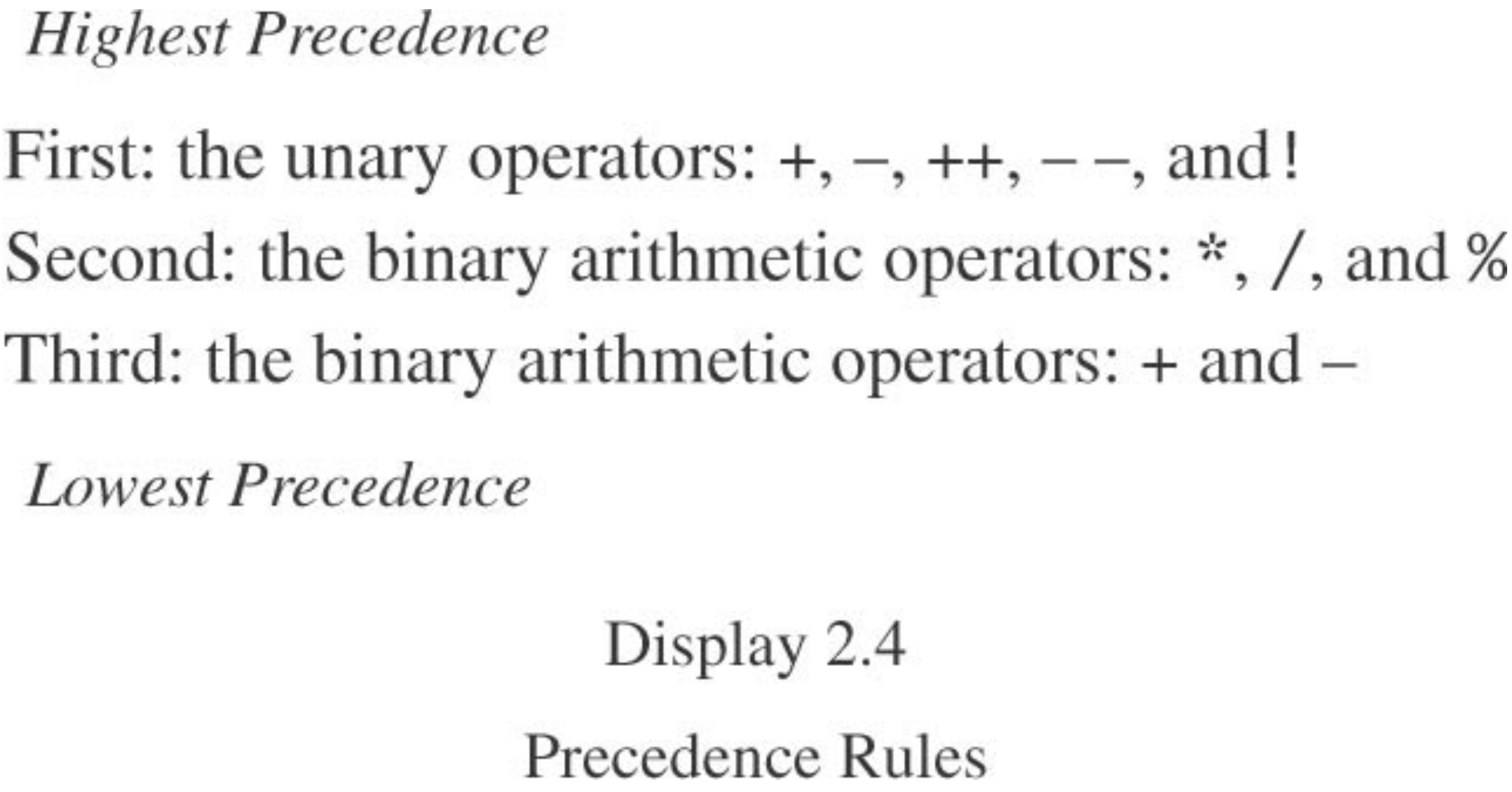
- Symbols that perform simple computations
- Assignment: =
- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Mod: %

# Order of Operations

- Follows standard math rules:
  - Parentheses
  - Multiplication and division
  - Addition and subtraction

  - double x = 3 / 2 + 1; // x = 2.0
  - double y = 3 / (2 + 1); // y = 1.0

# Order of Operations – Cont.



*Highest Precedence*

First: the unary operators: +, –, ++, – –, and !

Second: the binary arithmetic operators: *, /, and %

Third: the binary arithmetic operators: + and –

*Lowest Precedence*

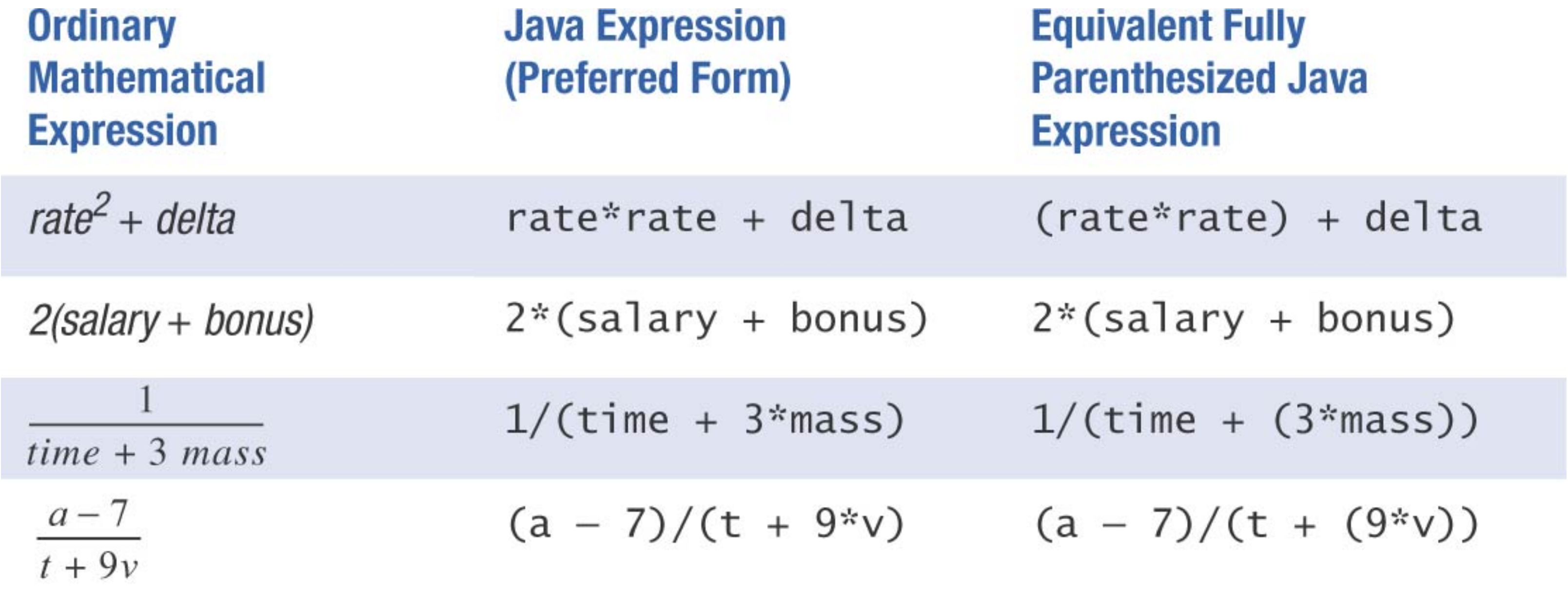
Display 2.4

Precedence Rules

# Order of Operations – Cont.

- The *binary* arithmetic operators `*`, `/`, and `%`, have *lower precedence* than the *unary* operators `++`, `--`, and `!`, but have *higher precedence* than the binary arithmetic operators `+` and `-`.
- When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.

# Sample Expressions



| Ordinary Mathematical Expression | Java Expression (Preferred Form) | Equivalent Fully Parenthesized Java Expression |
|---|---|---|
| $rate^2 + delta$ | rate*rate + delta | (rate*rate) + delta |
| $2(salary + bonus)$ | 2*(salary + bonus) | 2*(salary + bonus) |
| $\frac{1}{time + 3\ mass}$ | 1/(time + 3*mass) | 1/(time + (3*mass)) |
| $\frac{a - 7}{t + 9v}$ | (a - 7)/(t + 9*v) | (a - 7)/(t + (9*v)) |

Display 2.5

Arithmetic Expressions in Java

# Increment (and Decrement) Operators

- Used to increase (or decrease) the value of a variable by 1
- Easy to use, important to recognize
- The increment operator

  `count++` or `++count`

- The decrement operator

  `count--` or `--count`

# Increment (and Decrement) Operators

- equivalent operations

```
count++;
++count;
count = count + 1;

count--;
--count;
count = count - 1;
```

# Examples

- int k=0, y=0, x;
- x = ++k-y;
- System.out.println("x's value : "+x);

- int k=0, y=0, x;
- x = k++-y;
- System.out.println("x's value : "+x);

# Increment (and Decrement) Operators in Expressions

- after executing

  ```
  int m = 4;
  int result = 3 * (++m)
  ```

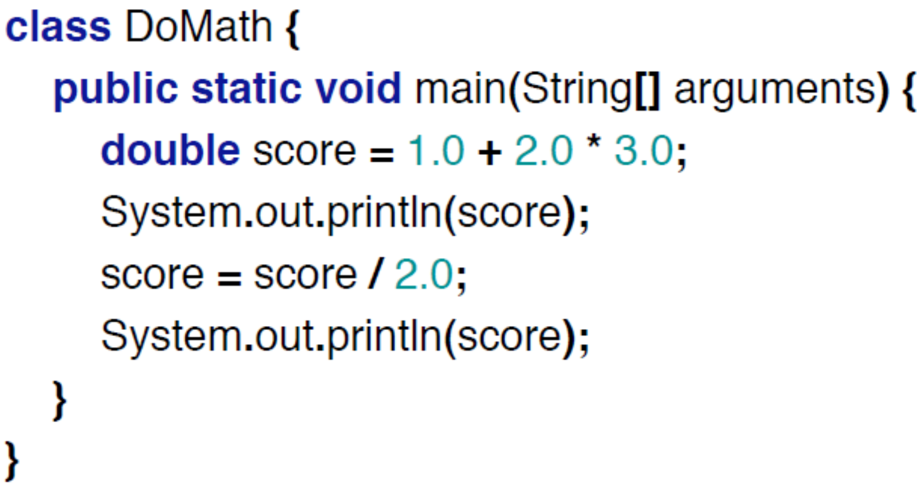
  `result` has a value of `15` and `m` has a value of `5`

- after executing

  ```
  int m = 4;
  int result = 3 * (m++)
  ```

  result has a value of `12` and `m` has a value of `5`

# Sample code: operators and assignments



```
class DoMath {
  public static void main(String[] arguments) {
    double score = 1.0 + 2.0 * 3.0;
    System.out.println(score);
    score = score / 2.0;
    System.out.println(score);
  }
}
```

```
class GravityCalculator {
 public static void main(String[] args) {
    double gravity = -9.81;
    double initialVelocity = 0.0;
    double fallingTime = 10.0;
    double initialPosition = 0.0;
    double finalPosition = .5 * gravity * fallingTime *
                                   fallingTime;
    finalPosition = finalPosition +
                     initialVelocity * fallingTime;
    finalPosition = finalPosition + initialPosition;
    System.out.println("An object's position after " +
     fallingTime + " seconds is " +
     finalPosition + “ m.");
      }
}
```

# Division

- Division (“/”) operates differently on integers and on doubles!
- Example:
  - double a = 5.0/2.0; // a = 2.5
  - int b = 4/2; // b = 2
  - int c = 5/2; // c = 2
  - double d = 5/2; // d = 2.0

# Conversion by casting

- int a = 2; // a = 2
- double a = 2; // a = 2.0 (Implicit)

- int a = 18.7; // ERROR
- int a = (int)18.7; // a = 18

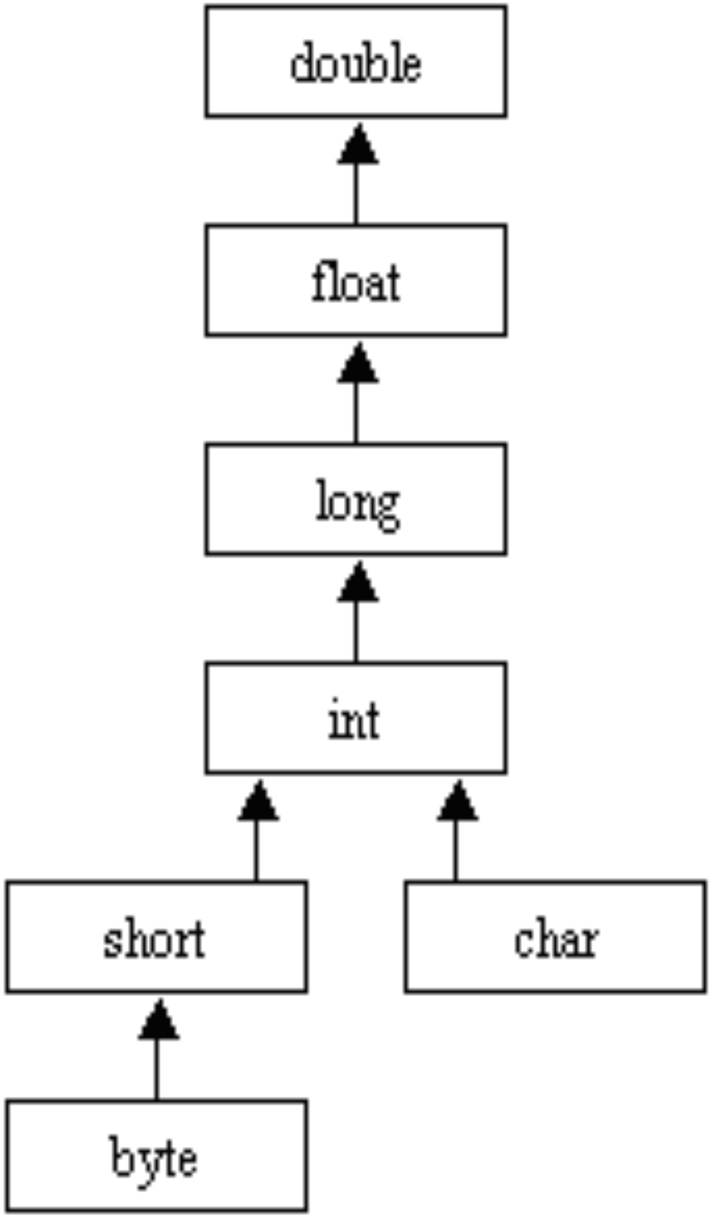
- double a = 2/3; // a = 0.0
- double a = (double)2/3; // a = 0.6666…

# Conversion by casting - Cont

- `double z = 3.0/2.0;`
- `System.out.println("===== "+z);`
-
- `double t = 3/2;`
- `System.out.println("===== "+t);`
-
- `double m = (double)3/2;`
- `System.out.println("===== "+m);`

# Casting

# Data Types and Their Relations in a Tree



# Casting example

```
•   public class Casting {
•           public static void main(String[] args){
•                   float a=12.5f;
•                   int i = (int) a;
•                   System.out.println("(int)12.5f==" + i);
•                    float f =  i;
•                   System.out.println("float değeri: " + f);
•                   System.out.print(f);
•                   f =f * i;
•                  System.out.println(f+"*" + i + "==" + f);
•           }
•   }
```

```
(int)12.5f==12
float değeri: 12.0
12.0*12==144.0
```

# Which ones are correct?

- float f = 2.34f;
- double d = f;
-
- f=d;
- d=f;
-
- long a = 15878;
- f = 1.1*a;
-
- int a = 78;
- long b = a*9876;

- byte a = 126;
- int b = ++a;
-
- byte a; int b;
- a=b;
-
- byte a = 1;
- short b = a;
-
- float f = 2.34f;
- char c=65;
-
- double d;
- char c=65;
- d = c*f*1.5;

```
• public class Casting_2 {
•     public static void main(String args[]) {
•                   byte x = 126;
•                   System.out.println( DoIt(x) );
•        }
•        static String DoIt(int a) {
•                   return "I've received an int of value "+a;
•        }
•        static String DoIt(byte a) {
•                   return "I've received a byte of value "+a;
•        }
• }
• public class Casting_3 {
•     public static void main(String args[]) {
•               char x = 'A';
•               System.out.println( DoIt(x) );
•      }
•      static String DoIt(int a) {
•               return "I've received an int of value "+a;
•      }
•      static String DoIt(byte a) {
•               return "I've received a byte of value "+a;
•      }
• }
```