# Data Conversion & Scanner Class

**August 29, 2007**

*ComS 207: Programming I (in Java)*
*Iowa State University, FALL 2007*
*Instructor: Alexander Stoytchev*



## Quick review of last lecture



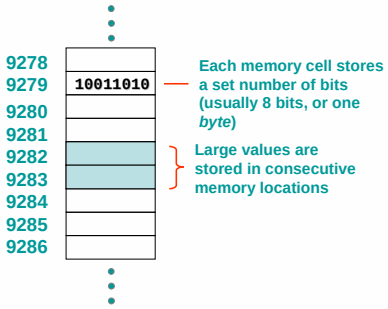
## Numeric Primitive Data

- **The difference between the various numeric primitive types is their size, and therefore the values they can store:**

| Type | Storage | Min Value | Max Value |
|---|---|---|---|
| `byte` | 8 bits | -128 | 127 |
| `short` | 16 bits | -32,768 | 32,767 |
| `int` | 32 bits | -2,147,483,648 | 2,147,483,647 |
| `long` | 64 bits | < -9 x $10^{18}$ | > 9 x $10^{18}$ |
| `float` | 32 bits | +/- 3.4 x $10^{38}$ with 7 significant digits | |
| `double` | 64 bits | +/- 1.7 x $10^{308}$ with 15 significant digits | |

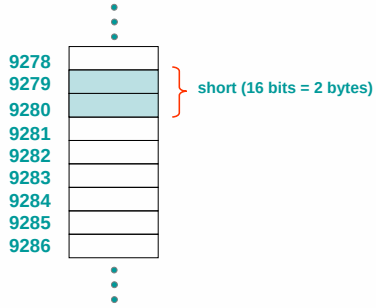


## Storing Information



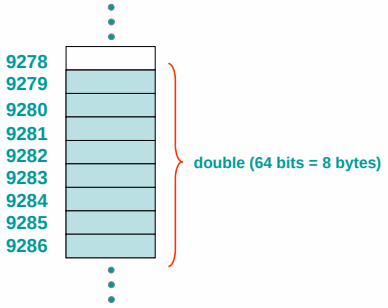



## Storing a short






## Storing a double



## Operator Precedence

| Precedence Level | Operator | Operation | Associates |
|---|---|---|---|
| 1 | + | unary plus | R to L |
| | – | unary minus | |
| 2 | * | multiplication | L to R |
| | / | division | |
| | % | remainder | |
| 3 | + | addition | L to R |
| | – | subtraction | |
| | + | string concatenation | |
| 4 | = | assignment | R to L |

FIGURE 2.4 Precedence among some of the Java operators



## Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
           4       1   3     2
```

Then the result is stored in the variable on the left hand side



## Other material from Sec 2.4



## Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of count

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

Then the result is stored back into count (overwriting the original value)



## Increment and Decrement

- The increment and decrement operators use only one operand
- The *increment operator* (++) adds one to its operand
- The *decrement operator* (--) subtracts one from its operand
- The statement

```
count++;
```

is functionally equivalent to

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



## Increment and Decrement

- The increment and decrement operators can be applied in *postfix form*:

```
count++
```

- or *prefix form*:

```
++count
```

- When used as part of a larger expression, the two forms can have different effects
- Because of their subtleties, the increment and decrement operators should be used with care

## Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```



## Assignment Operators

- There are many assignment operators in Java, including the following:

| Operator | Example | Equivalent To |
|---|---|---|
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |



## Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore

```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```



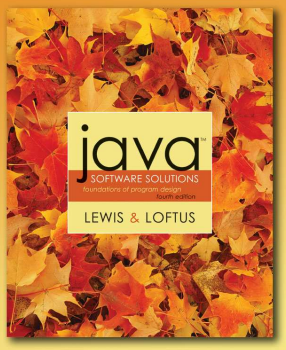
## Assignment Operators

- The behavior of some assignment operators depends on the types of the operands
- If the operands to the += operator are strings, the assignment operator performs string concatenation
- The behavior of an assignment operator (+=) is always consistent with the behavior of the corresponding operator (+)



Chapter 2

Sections 2.5 & 2.6




PEARSON Addison Wesley



## 2.5 Data Conversion

## Widening Conversions

| From | To |
|---|---|
| byte | short, int, long, float, or double |
| short | int, long, float, or double |
| char | int, long, float, or double |
| int | long, float, or double |
| long | float or double |
| float | double |

FIGURE 2.5 Java widening conversions



## Narrowing Conversions

| From | To |
|---|---|
| byte | char |
| short | byte or char |
| char | byte or short |
| int | byte, short, or char |
| long | byte, short, char, or int |
| float | byte, short, char, int, or long |
| double | byte, short, char, int, long, or float |

FIGURE 2.6 Java narrowing conversions



## Conversion Techniques

- **1) Assignment conversion**
  - **Value of one type is assigned to a variable of another type during which the value is converted to the new type.**
- **2) Promotion**
  - **Occurs automatically when certain operators need to modify their operands.**
- **3) Casting (a.k.a. type casting)**
  - **Specified explicitly by the programmer**



## Assignment conversion

```
float money;
int dollars;

dollars=5;

money = dollars;  // OK, money is now equal to 5.0

dollars= money;   //Compile error
```



## (automatic) promotion

```
float sum, result;
int count;

sum= 12.0;
count=5;

result = sum/count;  // count promoted to float
                     // before the division
```



## (automatic) promotion

```
// the number '5' is first promoted to a string and then
// the two strings are concatenated

System.out.println("Five is equal to " + 5);
```

## Type Casting

- The programmer explicitly asks the compiler to change the type of a variable or a temporary result before the next operation will take place.
- Without the cast Java typically will refuse to compile the program



## Type Casting

```
float money;
int dollars;

dollars=5;

money = dollars;  // OK, money is now equal to 5.0

dollars= (int) money;   //Compile error OK
```



## Type Casting + Promotion

```
float result;
int total, count;

total= 12;
count=5;

result =  (float) total / count;        //  result = 2.4
// 1. total is cast to float
// 2. count is promoted to float
// 3. the division is performed
```



## Type Casting + Promotion

```
float result;
int total, count;

total= 12;
count=5;

result =  (float) (total / count);   //  result = 2.0
// 1. total and count a divided using integer division
// 2. the intermediary result is cast to a float
// 3. this float value is assigned to result
```



## Casting Example



## Interactive Programs

- Programs generally need input on which to operate
- The `Scanner` class provides convenient methods for reading input values of various types
- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object

## Reading Input

- The following line creates a Scanner object that reads from the keyboard:

```
Scanner scan = new Scanner (System.in);
```

- The new operator creates the `Scanner` object
- Once created, the `Scanner` object can be used to invoke various input methods, such as:

```
answer = scan.nextLine();
```

- In order to use the `Scanner` object you must put this line at the top of your Java program

```
import java.util.Scanner;
```



## Reading Input

- The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used
- See Echo.java (page 91)
- The `nextLine` method reads all of the input until the end of the line is found
- The details of object creation and class libraries are discussed further in Chapter 3



## Input Tokens

- Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input
- White space includes space characters, tabs, new line characters
- The `next` method of the `Scanner` class reads the next input token and returns it as a string
- Methods such as `nextInt` and `nextDouble` read data of particular types
- See GasMileage.java (page 92)



## Scanner Class

```
Scanner (InputStream source)
Scanner (File source)
Scanner (String source)
    Constructors: sets up the new scanner to scan values from the specified source.

String next()
    Returns the next input token as a character string.

String nextLine()
    Returns all input remaining on the current line as a character string.

boolean nextBoolean()
byte nextByte()
double nextDouble()
float nextFloat()
int nextInt()
long nextLong()
short nextShort()
    Returns the next input token as the indicated type. Throws
    InputMismatchException if the next token is inconsistent with the type.

boolean hasNext()
    Returns true if the scanner has another token in its input.

Scanner useDelimiter (String pattern)
Scanner useDelimiter (Pattern pattern)
    Sets the scanner's delimiting pattern.

Pattern delimiter()
    Returns the pattern the scanner is currently using to match delimiters.

String findInLine (String pattern)
String findInLine (Pattern pattern)
    Attempts to find the next occurrence of the specified pattern, ignoring delimiters.
```

FIGURE 2.7 Some methods of the Scanner class



## More Scanner Examples



## THE END