Casting

- Explicit casting changes the data type of the *value* for a single use of the variable
- Precede the variable name with the new data type in parentheses:

  `(<data type>) variableName`

  » The type is changed to <data type> only for the single use of the returned value where it is cast.

- For example:

```
int n;
double x = 2.0;
n = (int)x
```

the value of **`x`** is converted from double to integer before assigning the value to **`n`**

Explicit casting is required to assign a higher type to a lower

- ILLEGAL: *Implicit* casting to a *lower* data type

```
int n;
double x = 2.1;
n = x;  //illegal in java
```

It is illegal since **x** is double, **n** is an int, and double is a higher data type than integer

data type hierarchy: **byte** ⇨ **short** ⇨ **int** ⇨ **long** ⇨ **float** ⇨ **double**

- LEGAL: *Explicit* casting to a lower data type **int n;**

```
double x = 2.1;
n = (int)x;  //legal in java
```

- You can always use an explicit cast where an implicit one will be done automatically, but it is not necessary

(예제)

```
public class EggBasket
{
   public static void main(String[] args)
   {
      int numberOfBaskets, eggsPerBasket, totalEggs;
int n;
double x = 2.1;
n = x;  //illegal in java

      eggsPerBasket = 6;
      numberOfBaskets = 10;
```

컴파일 결과

```
D:\My Documents\@@@@강의-자바-200601\@4판참고자료 및 3판 소스코드
\JAVA 4E\@21128-0131492217_src_4판소스코드\ch02\EggBasket.java:9:
possible loss of precision
found   : double
required: int

1 error

Tool completed with exit code 1
```

(예제)

```
public class EggBasketTest2
{
    public static void main(String[] args)
    {
        long n;
        float  x=0;
        n = x;
    }
}
```

```
public class EggBasketTest3
{
    public static void main(String[] args)
    {
        long n=0;
        float  x;
        x=n;
    }
}
```

컴파일 결과

```
D:\@@강의-201001-3-자바 프로그래밍\@v4\ch02\
possible loss of precision
found   : float
required: long
        n = x;
            ^
1 error

Tool completed with exit code 1
```

Characters as Integers

- Characters are actually stored as [redacted] according to a special code
  - » each printable character (letter, number, punctuation mark, space, and tab) is assigned a different integer code
  - » the codes are different for upper and lower case
  - » for example 97 may be the integer value for ‘a’ and 65 for ‘A’
- ASCII (Appendix 3) and Unicode are common character codes
- Unicode includes all the ASCII codes plus additional ones for languages with an alphabet other than English
- Java uses [redacted]

Casting a **`char`** to an **`int`**

- Casting a char value to int produces the ASCII/Unicode value
- For example, what would the following display?

```
char answer = `y`;
System.out.println(answer);
System.out.println((int)answer);
```

- Answer: the letter ‘y’ on one line followed by the ASCII code for ‘y’ (lower case) on the next line:

```
>y
>89
>
```

- **`‘7’ : 55`**

Assigning Initial Values to Variables

- Initial values *may or may not* be assigned when variables are declared:

```
//These are not initialized when declared
//and have unknown values
int totalEggs, numberOfBaskets, eggsPerBasket;

//These are initialized to 0 when declared
int totalEggs = 0;
int numberOfBaskets = 0;
int eggsPerBasket = 0;
```

- Programming tip: it is good programming practice always to initialize variables.

GOTCHA: Imprecision of Floating Point Numbers

- Computers store numbers using a fixed number of bits, so not every real (floating point) number can be encoded precisely
  - » an infinite number of bits would be required to precisely represent any real number
- For example, if a computer can represent up to 10 decimal digits, the number 2.5 may be stored as 2.499999999 if that is the closest it can come to 2.5
- Integers, on the other hand, are encoded precisely
  - » if the value 2 is assigned to an int variable, its value is precisely 2
- This is important in programming situations you will see later in the course

Arithmetic Operators

- addition (+), subtraction (-), multiplication (*), division (/)
- can be performed with numbers of any integer type, floating-point type, or combination of types
- result will be the [redacted] type that is in the expression
- Example:

```
amount - adjustment
```

  » result will be **`int`** if both **`amount`** and **`adjustment`** are **`int`**
  » result will be float if amount is **`int`** and adjustment is **`float`**

data type hierarchy: **byte** ⇨ **short** ⇨ **int** ⇨ **long** ⇨ **float** ⇨ **double**

[redacted] When Casting a **`double`** to an Integer

- Converting (casting) a double to integer does <u>not</u> round; it [redacted]
  - » the fractional part is lost (discarded, ignored, thrown away)
- For example:

```
int n;
double x = 2.99999;
n = (int)x;
```

  - » **<u>the value of `n` is now 2 ([redacted]d value of `x`)</u>**
  - » the cast is required
- This behavior is useful for some calculations, as demonstrated in *Case Study: Vending Machine Change*

Truncation When Doing Integer Division

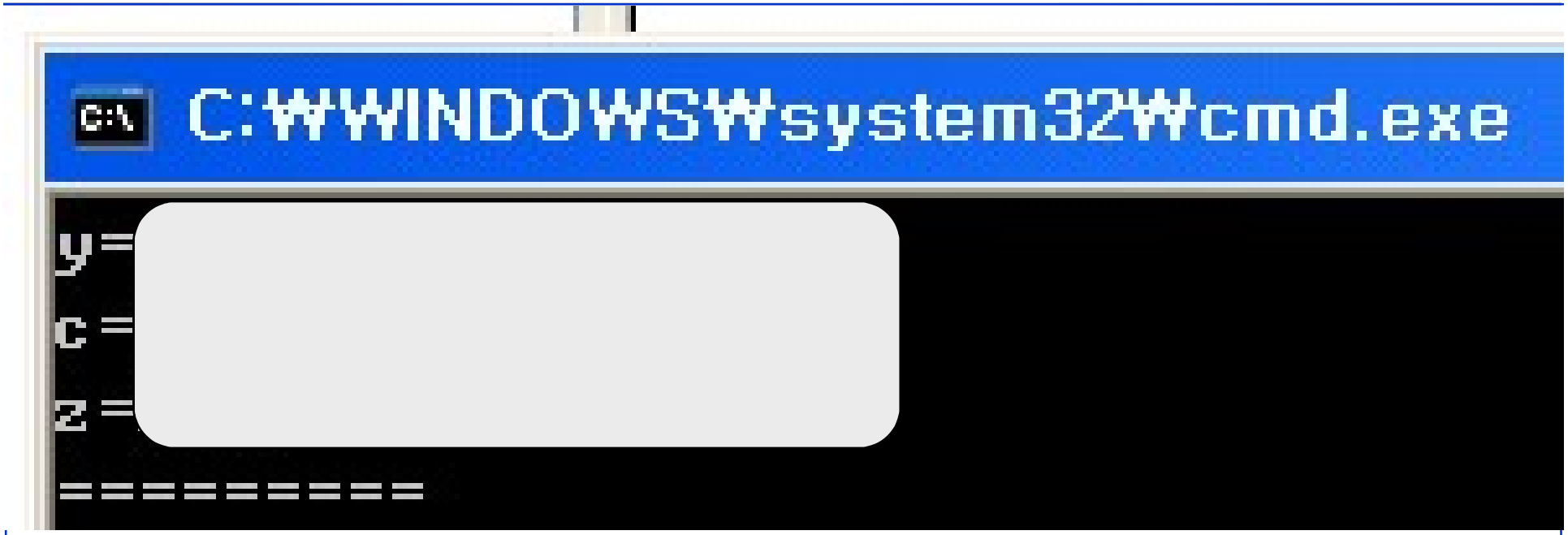
- No truncation occurs if at least one of the values in a division is type **`float`** or **`double`** (all values are promoted to the highest data type).
- Truncation occurs if *all* the values in a division are

(예제)

```
public class EggBasket
{
    public static void main(String[] args)
    {
int a = 4, b =5, c;
double x = 1.5, y,z;
y = b/x;//value returned by b is cast to double
        //value of y is
c = b/a;//all values are ints so the division

z = b/a;//all values are ints so the division
        //
System.out.println("y="+y);
System.out.println("c="+c);
System.out.println("z="+z);
System.out.println("=========");
```

실행결과



The Modulo Operator: `a % b`

- Used with integer types
- Returns the [illegible] of the division of `b` by `a`
- For example:

```
int a = 57; b = 16, c;
c = a % b;
```

`c` now has the value 9, the remainder when 57 is divided by 16

- A very useful operation: see *Case Study: Vending Machine Change*

List 2.3 A Change-Making Program

Excerpt from the **`ChangeMaker.java`** program:

```
int amount, originalAmount,
            quarters, dimes, nickels, pennies;
. . . // code that gets amount from user not shown
originalAmount = amount;
quarters = amount/25;
amount = amount%25;
dimes = amount/10;
amount = amount%10;
nickels = amount/5;
amount = amount%5;
pennies = amount;
```

If `amount` is 90 then there 90/25 will be 3, so there are three quarters.

If `amount` is 90 then the remainder of 90/25 will be 15, so 15 cents change is made up of other coins.

```
// Listing 2.3

import java.util.Scanner;

public class ChangeMaker
{
   public static void main(String[] args)
   {
      int amount;
      int originalAmount;
      int quarters;
      int dimes;
      int nickels;
      int pennies;

      System.out.println("Enter a whole number from 1 to 99.");
      System.out.println("I will find a combination of coins");
      System.out.println("that equals that amount of change.");
```

```
        Scanner keyboard = new Scanner(System.in);
         amount = keyboard.nextInt( );

         originalAmount = amount;
         quarters = amount / 25;
         amount = amount % 25;
         dimes = amount / 10;
         amount = amount % 10;
         nickels = amount / 5;
         amount = amount % 5;
         pennies = amount;

         System.out.println(originalAmount +
                        " cents in coins can be given as:");
         System.out.println(quarters + " quarters");
         System.out.println(dimes + " dimes");
         System.out.println(nickels + " nickels and");
         System.out.println(pennies + " pennies");
    }
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
Enter a whole number from 1 to 99.
I will output a combination of coins
that equals that amount of change.
87
87 cents in coins can be given as:
3 quarters
1 dimes
0 nickels and
2 pennies
계속하려면 아무 키나 누르십시오 . . .
```

Arithmetic Operator Precedence and Parentheses

- Java expressions follow rules similar to real-number algebra.
- Use parentheses to force precedence.
- Do not clutter expressions with parentheses when the precedence is correct and obvious.

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

- Shorthand notation for common arithmetic operations on variables used for counting
- Some counters count up, some count down, but they are integer variables
- The counter can be incremented (or decremented) before or after using its current value

```
int count;
…
```

`++count` preincrement count: count = count + 1 before using it

`count++` postincrement count: count = count + 1 after using it

`--count` predecrement count: count = count -1 before using it

`count--` postdecrement count: count = count -1 after using it

Increment and Decrement Operator Examples

common code

```
int n = 3;
int m = 4;
int result;
```

What will be the value of `m` and result after each of these executes?

(a) `result = n * ++m;//preincrement m`

(b) `result = n * m++;//postincrement m`

(c) `result = n * --m;//predecrement m`

(d) `result = n * m--;//postdecrement m`

Answers to Increment/Decrement Operator Questions

(a) 1) `m = m + 1;//m = 4 + 1 = 5`
2) `result = n * m;//result = 3 * 5 = 15`

(b) 1) `result = n * m;//result = 3 * 4 = 12`
2) `m = m + 1;//m = 4 + 1 = 5`

(c) 1) `m = m - 1;//m = 4 - 1 = 3`
2) `result = n * m;//result = 3 * 3 = 9`

(b) 1) `result = n * m;//result = 3 * 4 = 12`
2) `m = m - 1;//m = 4 - 1 = 3`

끝