Chapter 2

Primitive Types and Simple I/O

- 2.1 Variables and Expressions
  - » Primitive Data types
  - » Strings: a class
  - » Assignment
  - » Expressions
- 2.2 The Class String
- 2.3 Keyboard and Screen I/O
- 2.4 Documentation & Style
- 2.5 Graphics Supplement

OBJECTIVES

- Become familiar with the Java data types used for numbers, characters, and similar simple data. These types are called [redacted] types
- Learn about the assignment statement and expressions
- Find out about the Java data type used for strings of characters and learn how to do simple string processing. This will also serve to familiarize you with the notation used for classes, methods, and objects
- Learn about simple keyboard input and screen output.

2.1 VARIABLES AND EXPRESSIONS

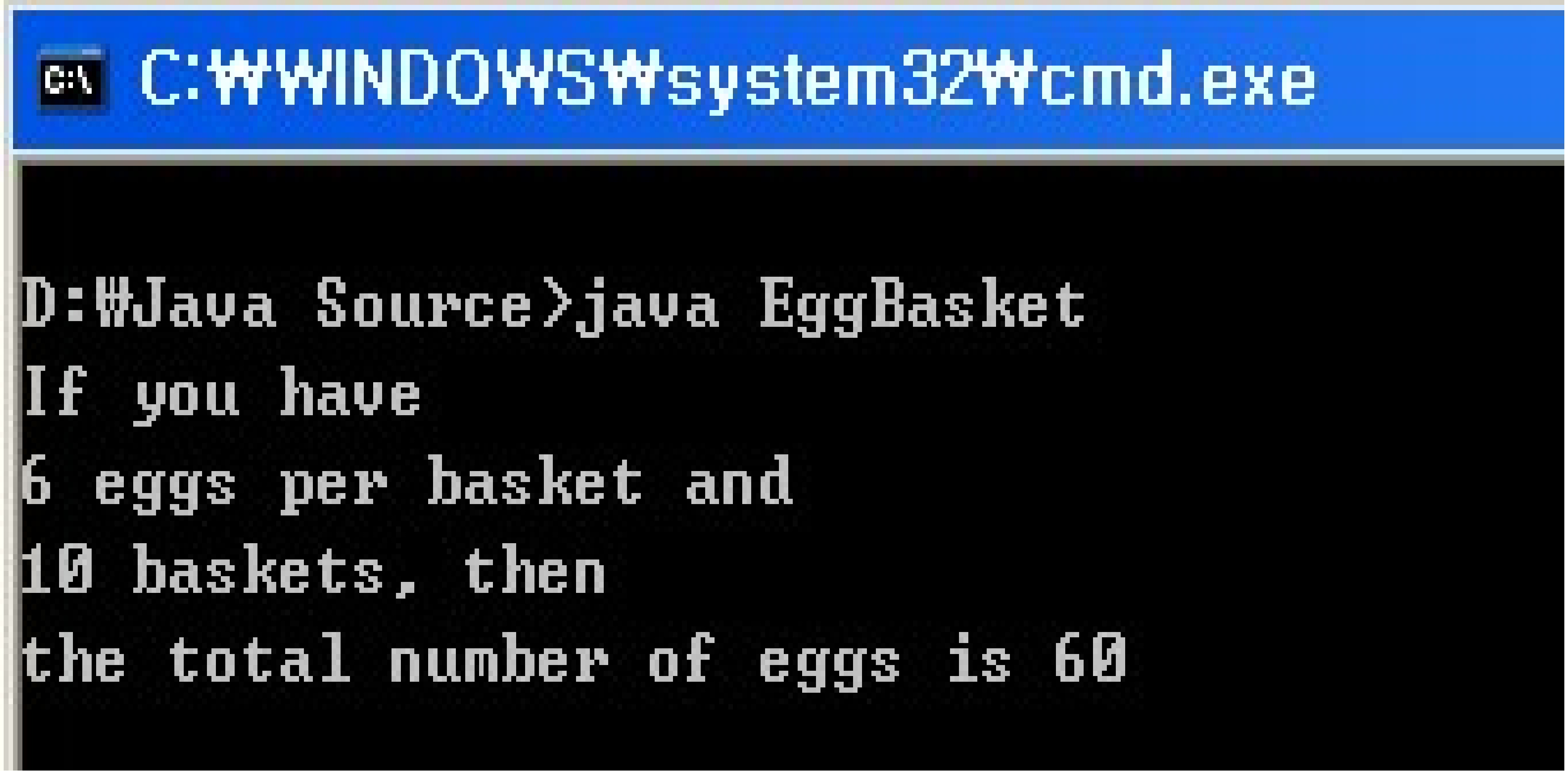
- Variable(??)
  - » Once a person has understood the way variables are used in programming, he has understood the quintessence of programming. – Dijkstra.
- Assignment (??)

What is a program variable?

- A [redacted] location to store data
  - a container for data
  - Value : the number, letter or other data item in a variable.
- It can hold only one type of data
  - for example only integers, only floating point (real) numbers, or only characters

```
// Listing 2.1

public class EggBasket
{
    public static void main (String [] args)
    {
        int numberOfBaskets, eggsPerBasket, totalEggs;
        numberOfBaskets = 10;
        eggsPerBasket = 6;
        totalEggs = numberOfBaskets * eggsPerBasket;
        System.out.println ("If you have");
        System.out.println (eggsPerBasket + " eggs per basket and");
        System.out.println (numberOfBaskets + " baskets, then");
        System.out.println ("the total number of eggs is " + totalEggs);
    }
}
```



```
D:₩Java Source>java EggBasket
If you have
6 eggs per basket and
10 baskets, then
the total number of eggs is 60
```

Creating Variables (declaring)

- All program variables **must be** [redacted] before using them

- A variable declaration associates a name with a storage location in memory and specifies the type of data it will store:

  Type Variable_1, Variable_2, …;

- For example, to create three integer variables to store the number of baskets, number of eggs per basket, and total number of eggs:

```
int numberOfBaskets, eggsPerBasket, totalEggs;
char answer;
double amount,interestRate;
```

- Location of declaration
  - » declared either just before it is used or at the start of a section of program that is enclosed in braces { }.

Changing the Value of a Variable

Usually a variable is changed (assigned a different value) somewhere in the program

- May be calculated from other values:

```
totalEggs = numberOfBaskets * eggsPerBasket;
```

- or read from keyboard input:

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

Two Main Kinds of *Types* in Java

[redacted] data types

- the simplest types
- cannot decompose into other types
- values only, no [redacted]
- Examples:
  `int` - integer
  `double` - floating point (real)
  `char` – character
- Begin with a [redacted] letter

[redacted] types

- Type for
- more complex
- composed of other types (primitive or class types)
- both [redacted]
- Examples:
  `SavitchIn`
  [redacted]
- **`Begin with a`** [redacted] **`letter`**

```
// Listing 1.1

import java.util.Scanner;

public class FirstProgram
{
    public static void main (String [] args)
    {
        System.out.println ("Hello out there.");
        System.out.println ("I will add two numbers for you.");
        System.out.println ("Enter two whole numbers on a line:");
        int n1, n2;
        Scanner keyboard = new Scanner (System.in);
        n1 = keyboard.nextInt ();
        n2 = keyboard.nextInt ();
        System.out.println ("The sum of those two numbers is");
        System.out.println (n1 + n2);
    }
}
```

Identifiers

- An [redacted] is the name of something (e.g. a variable, object, or method) used in a Java program.
- Syntax rules for identifiers tell what names are allowed.
- Naming conventions are not required by the compiler but are good practice.

Syntax Rules for Identifiers

Identifiers

- cannot be [redacted] words (e.g. “if,” “for”, etc.– see App. 1)
- must contain only letters, digits, and the underscore character, _.

$

- cannot have a digit for the first character.
  - » [redacted] is allowed but has special meaning, so do not use it.
- have no official [redacted] (there is always a finite limit, but it is very large and big enough for reasonable names)
- ***are case*** [redacted]
  - » **`junk`**, **`JUNK`**, and **`Junk`** are three valid and [redacted] identifiers, so be sure to be careful in your typing!
- Note that no [redacted] or [redacted] are allowed.

Naming Conventions

- Always use [redacted] names, e.g. **`finalExamScore`**, instead of something like `x`, or even just **`score`**.
- Use only letters and digits.
- [redacted] interior words in multi-word names, e.g. **`answerLetter`**.
- **Names of classes** start with an [redacted] letter.
  - » ***every program in Java is a class as well as a program***.
- Names of variables, objects, and methods start with a [redacted] letter.

Keyword??& reserved word??

- Keyword
  - » a word of a programming language that is special only [redacted]
  - » ex) FORTRAN

    REAL APPLE

    REAL = 3.4
- Reserved word
  - » a special word of a Programming language that [redacted] as a name.
  - » ex) Java .

Primitive Numeric Data Type

- integer—whole number

  examples: 0, 1, -1, 497, -6902

  » four data types: [redacted] **(1 bytes)**, [redacted] **(2 bytes)**, [redacted] **(4 bytes)**, [redacted] **(8 bytes)**

✓ floating-point number—includes fractional part

  examples: 9.99, 3.14159, -5.63, 5.0

  » Note: 5.0 is a floating-point number even though the fractional part happens to be zero.

  » two data types: [redacted] **(4 bytes)**, [redacted] **(8 bytes)**

The **`char`** Data Type

- The **`char`** data type stores a single “printable” character
  - » [redacted] quotes
  - » [redacted] bytes)

- For example:

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

prints (displays) the letter **`y`**

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* 9223372036854775807
`float`	*floating-point number*	*4 bytes*	$\pm 3.40282347 \times 10^{+38}$ *to* $\pm 1.40239846 \times 10^{-45}$
`double`	*floating-point number*	*8 bytes*	$\pm 1.76769313486231570 \times 10^{+308}$ *to* $\pm 4.94065645841246544 \times 10^{-324}$
`char`	*single character (Unicode)*	*2 bytes*	*all Unicode characters*
`boolean`	`true` *or* `false`	*1 bit*	*not applicable*

Display 2.2

Primitive Types

Assignment Statements

- most straightforward way to change value of a variable

  Variable = Expression

  ```
  answer = 42;
  ```

- = is [redacted] operator
- evaluate expression on right-hand side of the assignment operator
- variable on the left-hand side of the assignment operator gets expression value as new value

Type boolean

- true, false
- 1 bit

Assignment statement
Assignment Operator =

- The assignment operator is not the same as the equals sign in algebra.
- It means -

"Assign the value of the expression on the right side to the variable on the left side."

- Can have the same variable on both sides of the assignment operator:

```
int count = 10;// initialize counter to ten
count = count - 1;// decrement counter
```

new value of **count** = 10 - 1 = 9

Specialized Assignment Operators

- A shorthand notation for performing an operation on and assigning a new value to a variable
- General form: ***var <op>= expression;***
  - » equivalent to: ***var = var <op> (expression);***
  - » <op> is +, -, *, /, or %
- Examples:

```
amount += 5;
    //amount = amount + 5;

amount *= 1 + interestRate;
    //amount = amount * (1 + interestRate);
```

- Note that the right side is treated as a [redacted] (put parentheses around the entire expression)

Number constants

- Constant
  - » 2 (integer constant)
  - » 1.5 (floating point constant)
  - » 8.65e8(e notation, scientific notation, floating-point notation)
  - » ‘B’, ‘A’, ‘C’ (literals)
  - » value which [redacted] change.

Assignment Compatibility

- Can't put a square peg in a round hole
- Can't put a **`double`** value into an **`int`** variable
- In order to copy a value of one type to a variable of a different type, there must be a [redacted].
- Converting a value from one type to another is called [redacted]
- Two kinds of casting:
  - » automatic or [redacted] casting (??)
  - » [redacted] casting (??)

Returned Value

- Expressions *return* values: the number produced by an expression is “returned”, i.e. it is the “returned value.”

```
int numberOfBaskets, eggsPerBasket, totalEggs;
numberOfBaskets = 5;
eggsPerBasket = 8;
totalEggs = numberOfBaskets * eggsPerBasket;
```

  - » in the last line `numberOfBaskets` returns the value 5 and `eggsPerBasket` returns the value 8
  - » `numberOfBaskets * eggsPerBasket` is an expression that returns the integer value 40

- Similarly, methods return values

  `SavitchIn.readLine()` is a method that returns a string read from the keyboard

Casting: changing the data type of the *returned* value

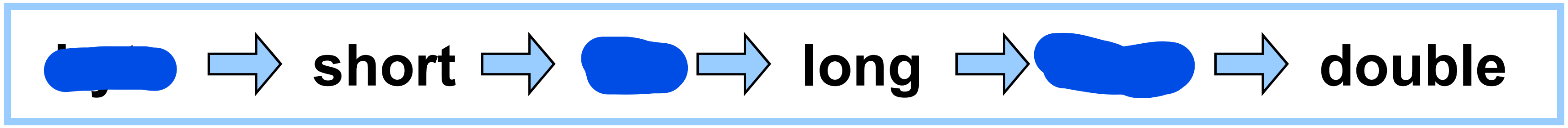
- Casting only changes the type of the [redacted] *value* (the single instance where the cast is done), not the type of the variable
- For example:

```
double x;
int n = 5;
x = n;
```

- Since **`n`** is an integer and **`x`** is a double, the value returned by **`n`** must be converted to type double before it is assigned to **`x`**

Implicit Casting

- Casting is done implicitly (automatically) when a “[redacted]” type is assigned to a “[redacted]” type
- The data type hierarchy (from lowest to highest):



[redacted] → **short** → **[redacted]** → **long** → **[redacted]** → **double**

- An **`int`** value will automatically be cast to a **`double`** value.
- A **`double`** value will **not** automatically be cast to an **`int`** value.

Implicit Casting Example: **`int`** to **`double`**

```
double x;
int n = 5;
x = n;
```

data type hierarchy:

byte ⇨ short ⇨ int ⇨ long ⇨ float ⇨ double

- the value returned by **`n`** is *cast* to a double,then assigned to **`x`**
- **`x`** contains 5.000… (as accurately as it can be encoded as a floating point number)
- This casting is done automatically because **`int`** is lower than **`double`** in the data type hierarchy
- The data type of the variable **`n`** is unchanged; is still an **`int`**

Data Types in an Expression: More Implicit Casting

- Some expressions have a mix of data types
- All values are automatically advanced (implicitly cast) to the [redacted] [redacted] the calculation
- For example:

```
double a;
int n = 2;
float x = 5.1;
double y = 1.33;
a = (n * x)/y;
```

`n` and `x` are automatically cast to type [redacted] before performing the multiplication and division

끝