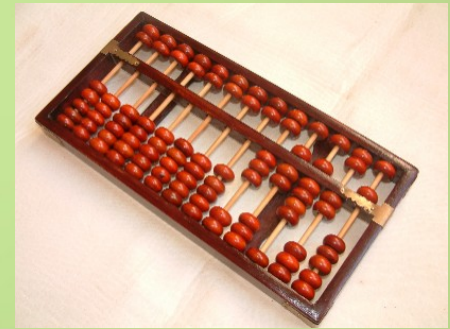


Lecture 2

Elementary Programming

- Primitive data types
- Identifiers and variables
- Assignment statements
- Arithmetic expressions



This image is from the [Wikimedia Commons](#).



香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

Example

- Problem Statement
 - Compute the final grade as a weighted sum of the scores obtained in exam, lab and homework
- Problem Analysis
 - Input:
 - Scores for Examination, lab and homework
 - Output:
 - Final grade
 - Additional information:
 - Pre-determined weights for examination, lab and homework



Example: Initial Algorithm

1. Determine the weights for each assessed components.
2. Obtain the input for exam, lab and homework scores.
3. Compute the final score:
 - 3.1 Compute the weighted exam score
 - 3.2 Compute the weighted lab score
 - 3.3 Compute the weighted homework score
 - 3.4 Compute the sum of the above weighted scores
4. Output the final score.



```
/**
 * CourseGrade determines the final grade which is computed as
 * the weighted sum of the grades obtained in exam, lab and homework
 */
public class CourseGrade
{
    public static void main(String[] args)
    {
        int examWeight = 70;    // Percentage weight given to examination
        int labWeight = 20;     // Percentage weight given to lab work
        int hwWeight = 10;      // Percentage weight given to homework assignment
        double examScore;       // Examination score obtained by student
        double labScore;        // Lab score obtained by student
        double hwScore;         // Homework score obtained by student
        double finalGrade;      // Final grade obtained by student
    }
}
```



```
// Ask student to input scores for exam, lab and homework
IO.output("Enter your exam grade: ");
examScore = IO.inputDouble( );
IO.output("Enter your lab grade: ");
labScore = IO.inputDouble( );
IO.output("Enter your homework grade: ");
hwScore = IO.inputDouble( );

// Computer final grade as the weighted sum of exam, lab and homework scores
examScore = examScore * (examWeight / 100.0);
labScore = labScore * (labWeight / 100.0);
hwScore = hwScore * (hwWeight / 100.0);
finalGrade = examScore + labScore + hwScore;

// Output the final grade
IO.outputln("Your final grade is " + finalGrade);

}

}
```



Identifiers

- An identifier is a sequence of characters consisted of
 - Letters (a-z, A-Z)
 - Underscores (_)
 - Dollar signs (\$)
 - Digits (0-9, the first character cannot be a digit)
- Identifiers are case-sensitive. For example: Hello, hello, WHOAMI, WhoAml, whoami are different identifiers.
- An identifier can be of any length, but cannot be one of the reserved words
 - (What is a reserved word?)

// valid identifiers

int examWeight = 10;

double examScore;

double exam_Score;

int perfect_score_100;

// Invalid identifiers

int perfect score 100; //WRONG

int today2-7-2014; //WRONG

int 2014y; // WRONG!

int int; // WRONG!



Reserved Words

- Example in our daily lives
 - Some vehicle registration plates are reserved for special purpose
 - Internet domain names, e.g. .gov, .edu
- Similarly, in Java, there are some reserved words



Examples of
“reserved words”
in our daily lives



Reserved words

Elementary programming	Branching and loops	Object and classes	Miscellaneous
boolean byte char double final float int long short void	break case continue default do else for if switch while	class instanceof new private protected public static this	import package return

This list is incomplete, but covers most of the reserved words in this course



Naming Conventions

- Meaningful names should be used
 - Avoid using x, y, z, var1, var2....
 - Uses meaningful names such as radius, center, area, score...
- **MixedCase** can be used for lengthy variables by separating words using upper case letters
 - Lower **camelCase**: starts with a lower case letter, e.g. variables and methods
 - Upper **CamelCase**: starts with upper case letter, e.g. classes



/ Try to avoid meaningless identifiers */*

int x;

double var1;

/ Meaningful identifiers with camelCase are preferable */*

double examScore;

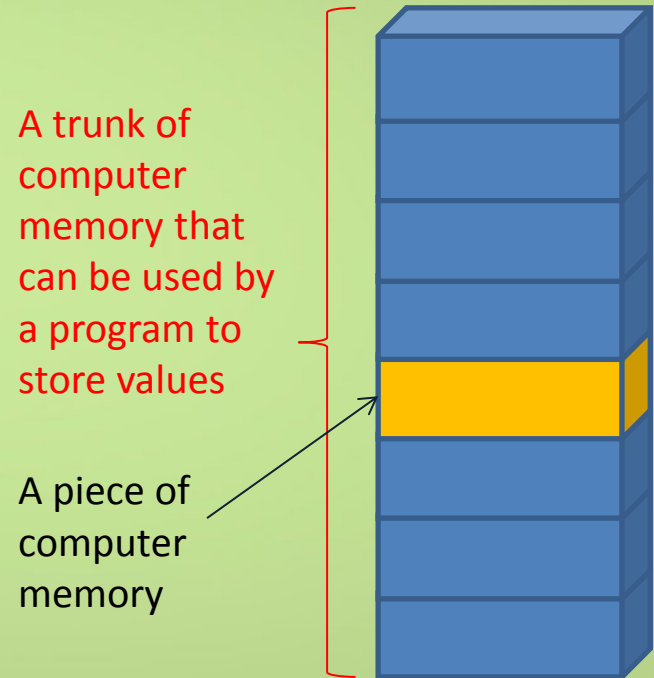
double areaOfCircle;

public **class** BankAccount



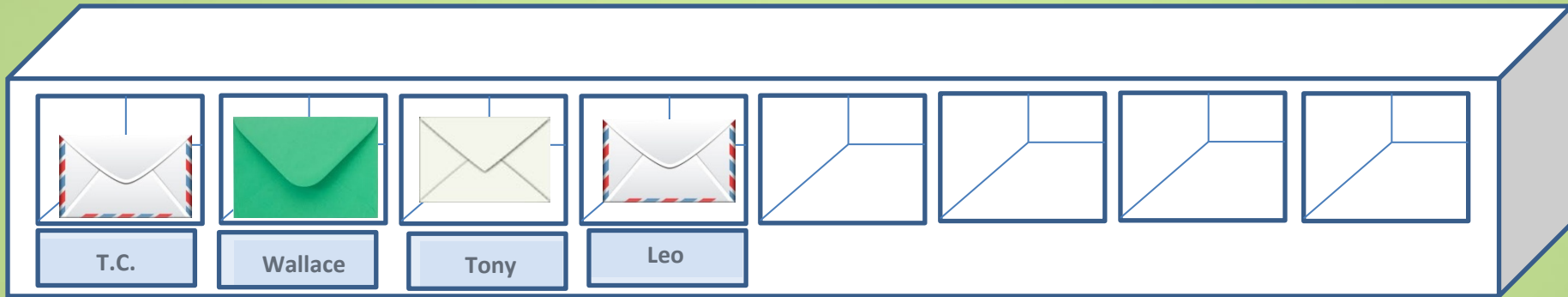
What is a variable?

- A **variable** is kind of identifier
- A variable represents a piece of **computer memory** that can store a value
- Typically such memory is requested by the program, and can be modified by the program
- The value of a variable can be changed so that a program would be more flexible for handling different inputs
- Examples: examScore, labScore, hwScore and finalGrade



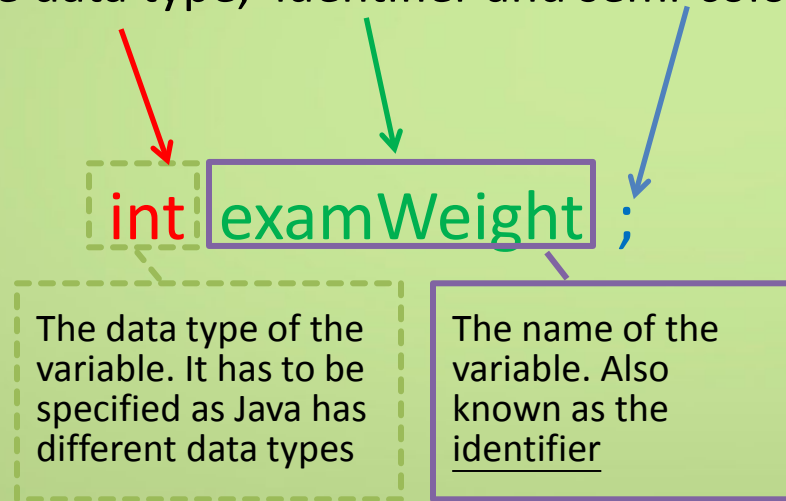
What is a variable?

An analogy is the use of mailbox, each mailbox is labeled by its owner (identifier) and different kinds of mails (values) can be put into the mailbox.



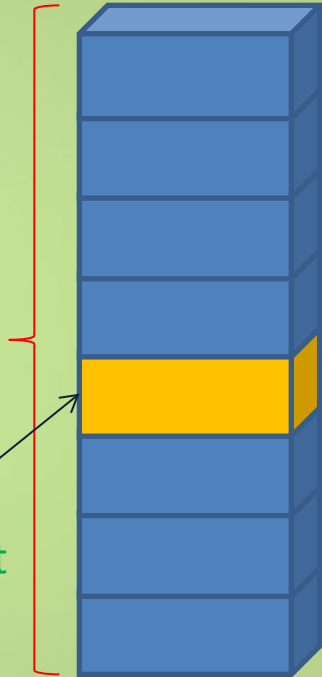
How is a Variable created?

- A variable is created through a declaration process
- A variable declaration in a program consists of **three** major parts
 - The data type, identifier and semi-colon



A trunk of computer memory that can be used by a Java program to store values

`examWeight`



```
/**
 * CourseGrade determines the final grade which is computed as
 * the weighted sum of the grades obtained in exam, lab and homework
 */
public class CourseGrade
{
    public static void main(String[] args)
    {
        int examWeight = 70; // Percentage weight given to examination
        int labWeight = 20; // Percentage weight given to lab work
        int hwWeight = 10; // Percentage weight given to homework assignment
        double examScore; // Examination score obtained by student
        double labScore; // Lab score obtained by student
        double hwScore; // Homework score obtained by student
        double finalGrade; // Final grade obtained by student
    }
}
```



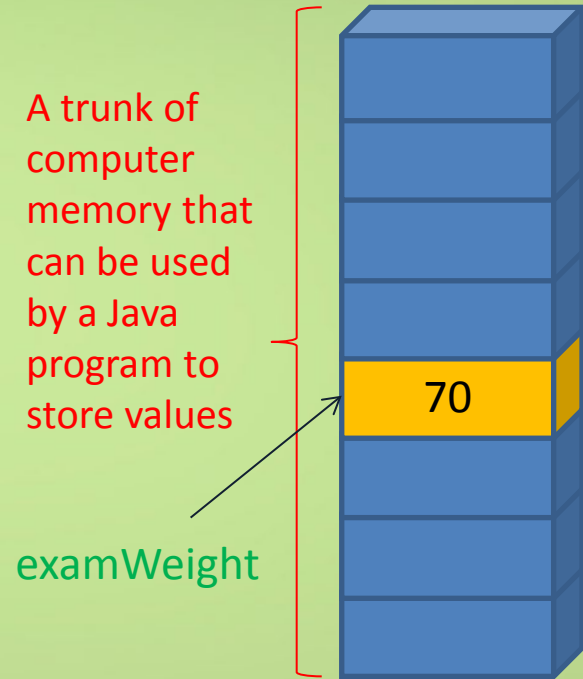
Initializing a variable

- After we have created a variable, we need to initialize the variable (i.e. give it an initial value):

```
int examWeight = 70;
```

- Declaration and initialization can be done in separate steps:

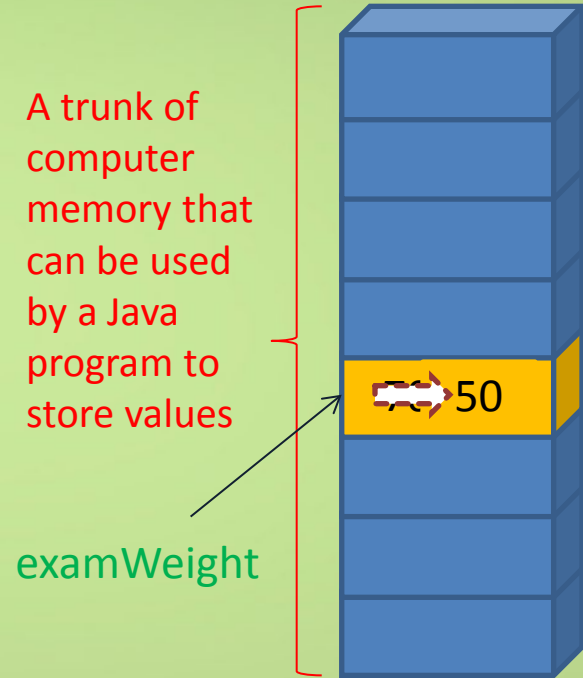
```
int examWeight ;  
examWeight = 70;
```



Changing the Value of a Variable

- After a variable is declared and initialized, its value can be modified by an assignment statement:

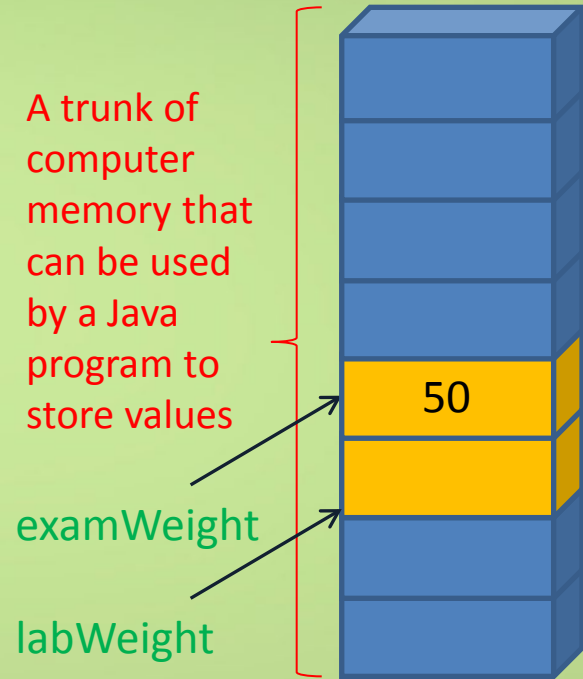
```
examWeight = 50;
```



Retrieve a Value from a variable

- The current value of a variable can be retrieved by referring to its name:

```
int labWeight = examWeight ;
```



Finalizing a variable's value

- Sometimes we want to keep a variable's value unchanged throughout the program, for example:
 - Mathematical constants, e.g. π ($\pi = 3.14159$)
 - examWeight, labWeight, hwWeight in the CourseGrade program
- We can prevent accidental changes to the value by “finalizing” a variable's value
 - Adding the keyword “final” when declaring the variable
 - Such variables are also referred to as **constants**.



Finalizing a variable's value

- We can assign value to a final variable only **once**

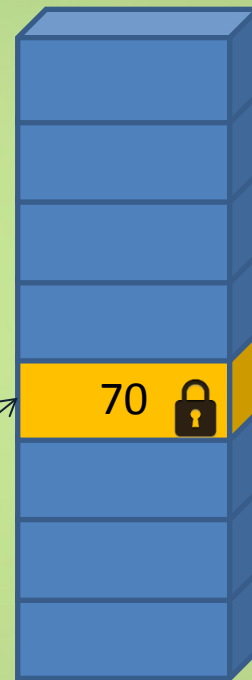
```
final int examWeight = 70;
```

- Re-assigning a final variable creates a compilation error

```
examWeight  = 50;
```

A trunk of computer memory that can be used by a Java program to store values

examWeight



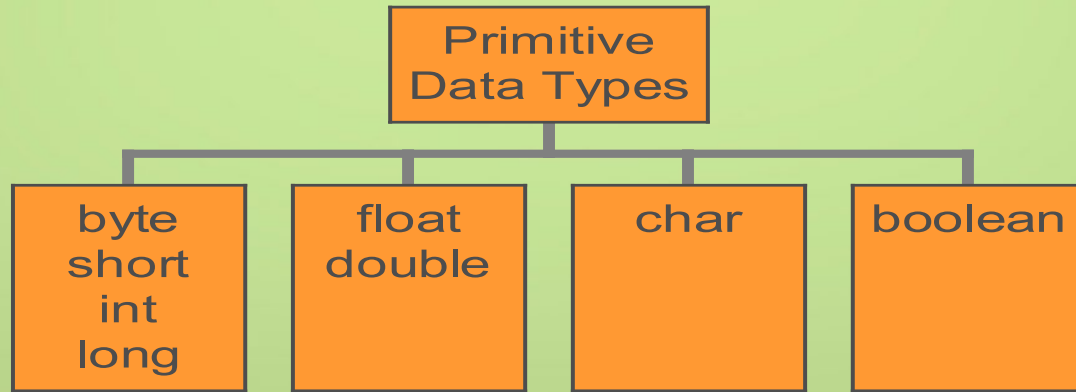
Introduction to Data Types

- Data types is a very important concept in high level programming language.
 - The way to define a variable is by specifying a type
 - All variables and names referred in a Java program must be defined before it can be used.
 - Each data type has its own properties and memory space requirements



Java Primitive Data Types

- A type defines a set of **values** and a set of **operations** that can be applied on those values.
- The set of values for each type is known as the **domain** for the type.
- Java supports eight primitive data types:



Java Primitive Data Types

- Integers
 - Data types
 - byte, short, int, long
 - Values stored are exact without any approximation
- Floating-point numbers
 - Data types
 - float, double
 - Values stored are approximated
- Booleans
 - Data types
 - boolean
 - There are only 2 possible values: **true** and **false**
- Characters
 - Data types
 - char
 - They are used to store symbols



Data types: Integers

Name	Range
byte	-2^7 to 2^7-1 (-128 to 127)
short	-2^{15} to $2^{15}-1$ (-32768 to 32767)
int	-2^{31} to $2^{31}-1$
long	-2^{63} to $2^{63}-1$

Advantages:

Represents the exact numerical value without approximation

Disadvantages:

Cannot represent floating-point values



Data types: floating-point numbers

(for reference **ONLY**, you are not required to memorize them)

Name	Range
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38
double	Negative range: -1.7976931348523157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348523157E+308

The range is much larger than integers (long) ($\sim 2^{63}$ only)



Expressions

- In writing programs, we use expressions as building blocks, which typically describe actions the program should take
- Two types of expressions
 - Arithmetic expressions
 - Boolean expressions
- Examples:
 - Computing final grade as a weighted sum of some scores
 - Check if a student received a passing grade



Arithmetic Expression

- An arithmetic expression is a sequence of variables, constants, literals connected by arithmetic operators
- Examples
 - An arithmetic expression “2+3” has two literals (2 and 3) connected by a ‘+’ operator
 - An arithmetic expression “celsius*9/5+32” has one variable (celsius), three literals (9, 5 and 32) connected by three operators (*, / and +)



Numeric literals

- A literal is a constant value that appears directly in the program
- There are two types of numeric literals
 - Integer literals
 - Floating-point literals
- A compilation error occurs if the literal is too large to fit into a variable

// Integer literals

byte aByte = 10;

int aInt = 10;

// floating point literals

float aFloat = 10.0f;

double aDouble = 10.0;

// Compilation error examples

float aFloat2 = 10.0; // without suffix "f"

byte aByte2 = 1000; // too large for byte



Arithmetic operators

Some commonly used arithmetic operators:

Operator	Examples	Result
(+) Addition	$2 + 3$	5
(-) Subtraction	$2 - 3$	-1
(*) Multiplication	$2 * 3$	6
(/) Division	2 / 3	0 (Integer division)
(%) Modulus Would work on floating point numbers too!	2 % 3	2 (The remainder of 2 / 3)



Example: Division and Modulus

$$\begin{array}{r} 34 \leftarrow (a/b) \\ b \rightarrow 3 \overline{) 104} \leftarrow a \\ \underline{102} \\ 2 \leftarrow (a \% b) \end{array}$$




Integer division and floating-point division

- For data types storing numerical values, we have integers (int) and floating point numbers (double)
- The results for integer and floating numbers are different on division:
 - Integer division
 - Division between two ints: the result is the integral part of the division, the decimal part of the number would be discarded.
 - Examples: $2/3$ results in 0, $3/2$ results in 1
 - Floating-point division
 - Division between one or two double's results in a double
 - Examples: $2/3.0$ results in 0.66666 (integer divided by double)
 $3.0/2$ results in 1.5 (double divided by integer)
 $10.0/2.0$ results in 5.0 (double divided by double)



Operators and Precedence

- Which of the following is it equivalent to $mx + b$?
 - $(m * x) + b$ 
 - $m * (x + b)$
- Operator precedence tells the order in which different operators in an expression are evaluated.
- Standard precedence order
 - $()$ Evaluated first, if nested then evaluate the innermost first.
 - $*$ $/$ $\%$ Evaluated second. If there are several, then evaluate from left-to-right.
 - $+$ $-$ Evaluate third. If there are several, then evaluate from left-to-right.



Associativity

- **Associativity** is used to determine the order in which operators with the same precedence are evaluated in an expression.

- Example:
$$3 * 8 / 4 \% 4 * 5$$

1 2 3 4
↓ ↓ ↓ ↓

is the same as

$$((((3 * 8) / 4) \% 4) * 5)$$

- **Parentheses** can be inserted if you are not sure about (or want to enforce) the evaluation order



Assignment statements

- Syntax of an assignment statement:
 - **Variable** = **Expression**;
 - It means to assign the value evaluated from an **expression** to the **variable**
 - The original value stored in that variable (if any) will be replaced

- Example:

```
int one = 1;    // variable one is initialized to 1
one = one + 1; // Now, the variable one has a value of 2
```



Type Conversions

- Type conversion changes the value of one data type to another.
- Type conversion can be done in two different ways: implicit and explicit
- Implicit conversion is done without any special directive from the programmer.
 - For examples, to assign an int to a float
 - Basic rule: The conversion is allowed if the range of values of the first type is a subset of the second (widening conversion).
- Explicit conversion is done by **type casting**.
 - The name of the target type, in parentheses, is placed in front of the data type to be converted.
 - Example:

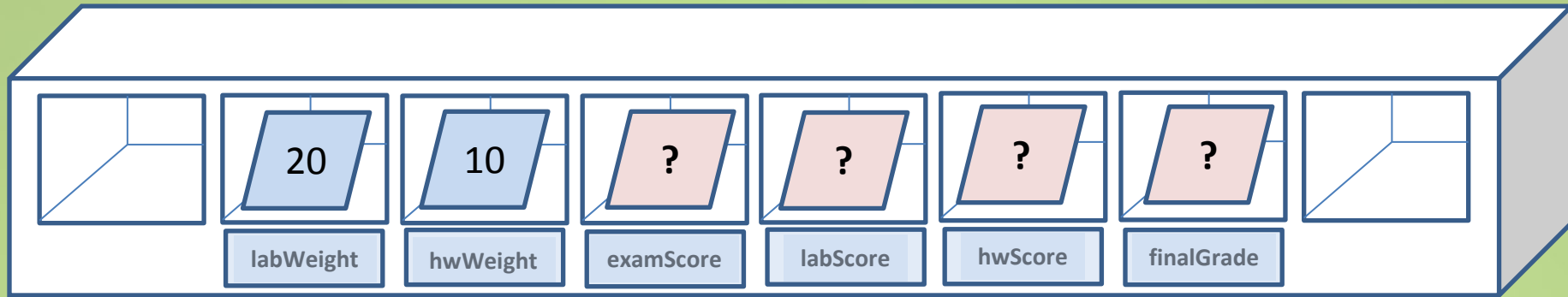
```
double dValue = 100.0;
int ivalue = (int) dValue;
float fValue = (float) dValue;
```



```
/**
 * CourseGrade determines the final grade which is computed as
 * the weighted sum of the grades obtained in exam, lab and homework
 */
public class CourseGrade
{
    public static void main(String[] args)
    {
        int examWeight = 70;    // Percentage weight given to examination
        int labWeight = 20;     // Percentage weight given to lab work
        int hwWeight = 10;      // Percentage weight given to homework assignment
        double examScore;       // Examination score obtained by student
        double labScore;        // Lab score obtained by student
        double hwScore;         // Homework score obtained by student
        double finalGrade;      // Final grade obtained by student
    }
}
```



Declaration of Data Type



examWeight

int examWeight = 70;

int labWeight = 20;

int hwWeight = 10;

double examScore;

double labScore;

double hwScore;

double finalGrade;



```
// Ask student to input scores for exam, lab and homework
IO.output("Enter your exam grade: ");
examScore = IO.inputDouble( );
IO.output("Enter your lab grade: ");
labScore = IO.inputDouble( );
IO.output("Enter your homework grade: ");
hwScore = IO.inputDouble( );

// Computer final grade as the weighted sum of exam, lab and homework scores
examScore = examScore * (examWeight / 100.0);
labScore = labScore * (labWeight / 100.0);
hwScore = hwScore * (hwWeight / 100.0);
finalGrade = examScore + labScore + hwScore;

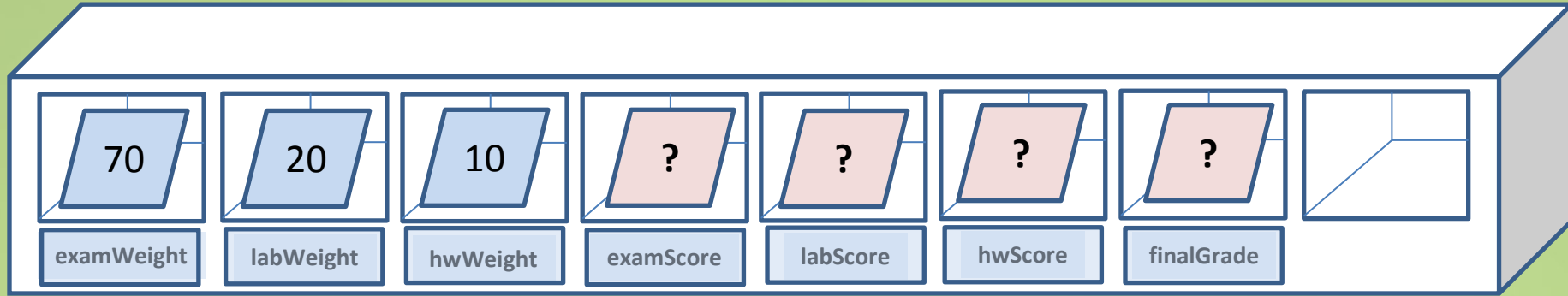
// Output the final grade
IO.outputln("Your final grade is " + finalGrade);

}

}
```



Assignment of Values to Variables



```
examScore = IO.inputDouble( ) ;
```

90.0

```
labScore = IO.inputDouble( ) ;
```

85.0

```
hwScore = IO.inputDouble( ) ;
```

80.5



```
// Ask student to input scores for exam, lab and homework
IO.output("Enter your exam grade: ");
examScore = IO.inputDouble( );
IO.output("Enter your lab grade: ");
labScore = IO.inputDouble( );
IO.output("Enter your homework grade: ");
hwScore = IO.inputDouble( );

// Computer final grade as the weighted sum of exam, lab and homework scores
examScore = examScore * (examWeight / 100.0);
labScore = labScore * (labWeight / 100.0);
hwScore = hwScore * (hwWeight / 100.0);
finalGrade = examScore + labScore + hwScore;

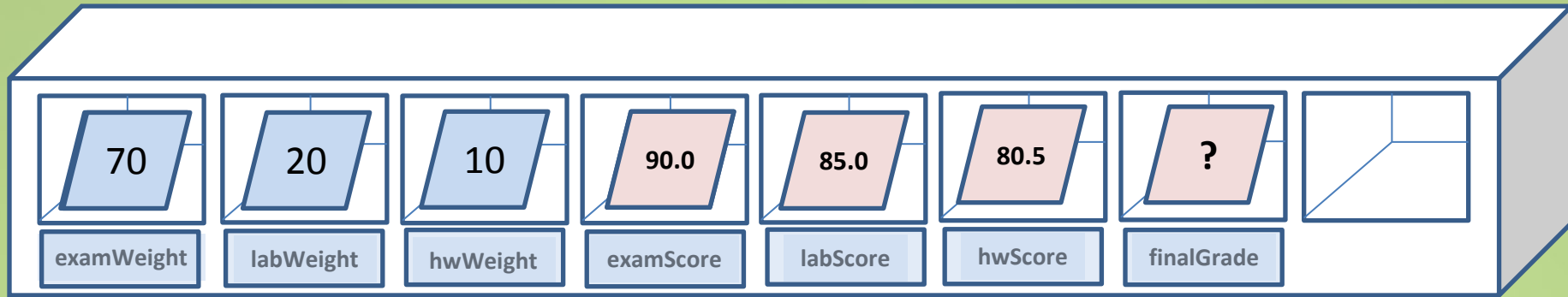
// Output the final grade
IO.outputln("Your final grade is " + finalGrade);

}

}
```



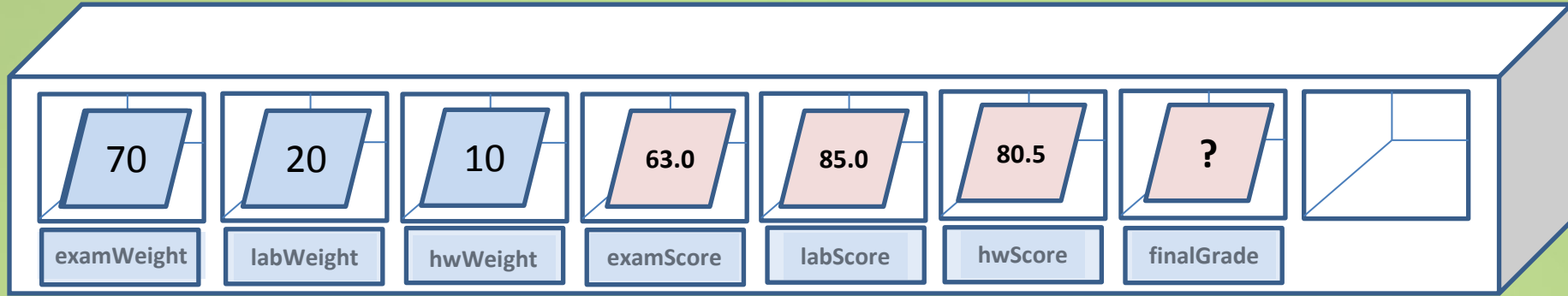
Changing value of a variable



$$\text{examScore} = \text{examScore} * (\text{examWeight} * 0.0);$$



Changing value of a variable



```
examScore = examScore * (examWeight / 100.0) ;
```

```
labScore = labScore * (labWeight / 100.0) ;
```

```
hwScore = hwScore * (hwWeight / 100.0) ;
```

```
finalGrade = examScore + labScore + hwScore ;
```

17.0

8.05

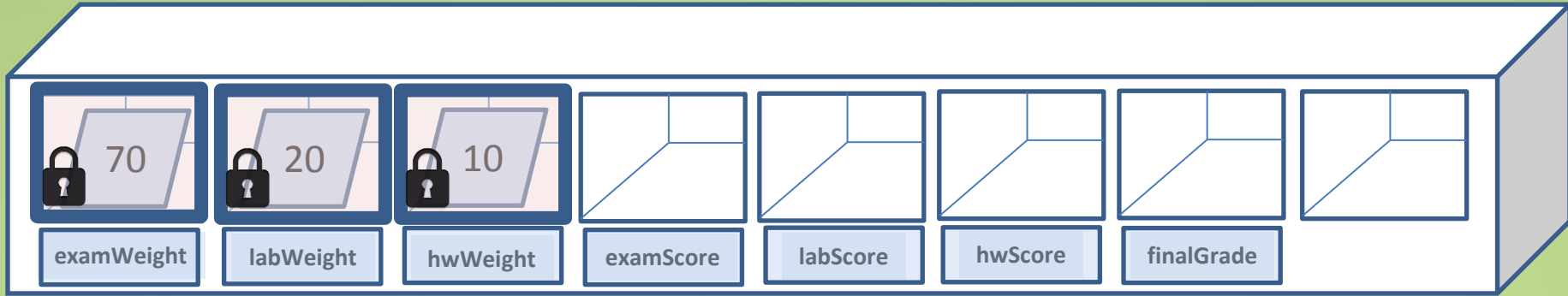
88.05




```
/**
 * CourseGrade determines the final grade which is computed as
 * the weighted sum of the grades obtained in exam, lab and homework
 */
public class CourseGrade
{
    public static void main(String[] args)
    {
        final int examWeight = 70;    // Percentage weight given to examination
        final int labWeight = 20;     // Percentage weight given to lab work
        final int hwWeight = 10;      // Percentage weight given to homework assignment
        double examScore;              // Examination score obtained by student
        double labScore;               // Lab score obtained by student
        double hwScore;                // Homework score obtained by student
        double finalGrade;             // Final grade obtained by student
    }
}
```



Declaration of Constants

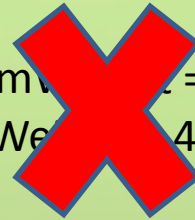


```
final int examWeight = 70;
```

```
final int labWeight = 20;
```

```
final int hwWeight = 10;
```

```
examWeight = 50;  
labWeight = 40;
```



Simple I/O

- A program interacts with users through input and output devices
- Console output: `System.out.println` and `System.out.print`
 - Example: `System.out.println("Hello, world!");`
- Console input using the `Scanner` class will be discussed later
- I/O class for output: `IO.outputln` and `IO.output`
 - Examples: `IO.output("Enter your exam grade: ");`
`IO.outputln("Your final grade is " + finalGrade);`
- I/O class for input: `IO.inputInteger` and `IO.inputDouble`
 - Examples: `double examScore = IO.inputDouble();`
`int intValue = IO.inputInteger();`
- Add the following statement at the beginning of your program:
 - `import comp102x.IO;`



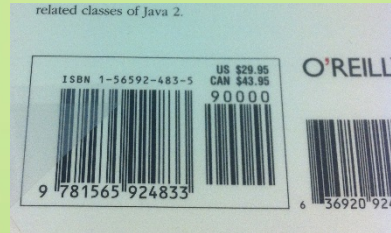
User Interface for Input

- Computers are not very good in reading handwritten/printed words or understanding speech
- Need user interface for inputting numbers and text to a computer
 - Keyboard
 - Visual input, e.g. barcode
 - Voice input, e.g. Siri



Barcode

- A barcode is a machine readable representation of data
- Linear (or 1-D) Barcodes represent data by varying the widths and spacings of a set of parallel lines



- Developed in the 1960s and became a commercial success due to its widespread use in automated checkout systems.



Barcode Example

```
import comp102x.IO;
// This program demonstrates simple arithmetic on values represented by barcodes
public class BarcodeDemo{
    public static void main(String[ ] args) {
        // Declare and initialize two long variables
        long value1 = IO.inputInteger(); long value1 = IO.inputBarcode(); // Read a barcode from an existing file
        long value2 = IO.inputInteger(); long value2 = IO.inputBarcode(); // Read another barcode from an existing file
        // Output the values represented by the two barcodes
        IO.outputln("The value of the 1st barcode is: " + value1);
        IO.outputln("The value of the 2nd barcode is: " + value2);
        // Add and multiply two barcodes
        long addResult = value1 + value2;
        long mulResult = value1 * value2;
        // Output the calculation results
        IO.outputln("The result of adding the two barcodes values is: " + addResult);
        IO.outputln("The result of multiplying the two barcodes values is: " + mulResult);
        // Output the calculation results as images on the file system
        IO.outputBarcode(addResult);
    }
```

