# Review

Variables (Primitive Data Types)

# Objectives

- To understand the purpose of ***variables***
- To understand how information is stored in computer's memory
- To be able to **define**, **initialize** and **use** ***variables*** and ***finals***
- To be able to assign values to ***variables***
- To understand the properties and limitations of different ***variable*** types (e.g integer and floating-point numbers)

# Variables: Properties

- A variable
  - Is used to store information ← contents of the variable
    - A variable can contain one piece of of information at a time.
  - Has an identifier ← name of the variable
    The programmer picks a good name
    - A good name describes the contents of the variable or what the variable will be used for
  - Has a location in memory ← not controlled by programmer
  - Has a scope and lifetime ← controlled by programmer

# Defining Variables – Type & Initialization /1

- When creating variables, the programmer **MUST** specify the type of information to be stored.
  - e.g., int, double, string
- The programmer **MAY** give a variable an initial value.
  - *Initialization* is putting a value into a variable when the variable is created.
  - Initialization is recommended not required.

**Example:**

```
int  age = 23;
string  name = "Judy";
double  salary = 3000.00;
```

# Defining Variables – Type & Initialization /2

- The value of a variable can be modified (re**assigned**), if necessary.

```
int age = 24;
float salary = 100;
salary = salary + 400.00;
```

- Several variables can be declared on the same line:

```
int diameter, area;
int day = 15, month = 1, year = 2010;
```

Separate them by comma

- Several variables can be **initialized** with one value:

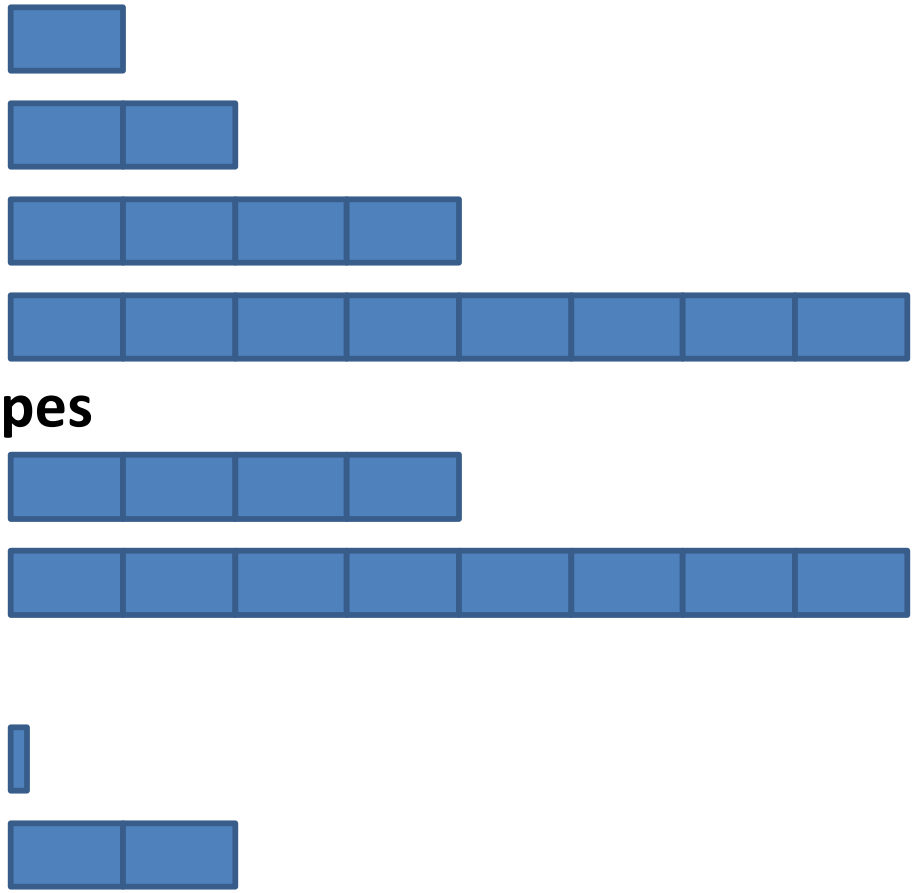
```
int x =2, y = x, w = y;
// the following line prints: 2, 2, 2
System.out.println(x, w, y);
```

# All Primitive Types

- **Integer Types**
  - `byte:` A very small number (-128 to +127)
  - `short:` A small number (-32768 to +32767)
  - `int:` A large number (-2,147,483,648 to +2,147,483,647)
  - `long:` A huge number
- **Floating Point Types**
  - `float:` A huge number with decimal places
  - `double:` Much more precise, for heavy math
- **Other Types**
  - `boolean:` `true` or `false`
  - `char:` One symbol in single quotes ‘a’

# Primitive Type Storage

- **Integer Types**
  - `byte:`
  - `short:`
  - `int:`
  - `long:`
- **Floating Point Types**
  - `float:`
  - `double:`
- **Other Types**
  - `boolean:`
  - `char:`



# Variable Naming Rules in Java

- An identifier in Java
  - Is a string of letters (a to z, or A to Z), underscore ( _ ) and numbers (0 to 9) only.
  - Can start either by a letter (a to z, or A to Z), or an underscore ( _ ).
- An identifier must be either one word or two or more words connected to each other by an underscore
- Identifiers have no semantics ← but better use meaningful words and avoid offensive words
- An identifier **cannot** have any of the following characters:

| | | | | | | |
|---|---|---|---|---|---|---|
| / | , | ; | . | & | * | |
| + | - | # | = | " | ' | @ |

# Variable Naming Rule in Java

- Variable names are case-sensitive, that is, `Apple` and `apple` are different names.
  - It is a good idea to use only lowercase letters for names.
  - It is a good idea to separate multiple word names by underscore (never use – or . )
- It is better if variables names be of a length less than 255 characters ← not a Java standards
- Reserved words (i.e. words reserved exclusively for their special Java meanings) such as `double`, `final` or `int` CANNOT be used as variable names.

# Data Types

| Number | Type | Comment |
|---|---|---|
| 6 | `int` | An integer has no fractional part. |
| –6 | `int` | Integers can be negative. |
| 0 | `int` | Zero is an integer. |
| 0.5 | `double` | A number with a fractional part has type `double`. |
| 1.0 | `double` | An integer with a fractional part .0 has type `double`. |
| 1E6 | `double` | A number in exponential notation: $1 \times 10^6$ or 1000000. Numbers in exponential notation always have type `double`. |
| 2.96E-2 | `double` | Negative exponent: $2.96 \times 10^{-2} =$ 2.96 / 100 = 0.0296 |
| 100,000 | | **Error**: Do not use a comma as a decimal separator. |
| 3 1/2 | | **Error**: Do not use fractions; use decimal notation: 3.5 |

# final

- What is a final?
  - It is an identifier that indicates a computer memory space, with an specific data type, such as: integer (whole number), double (real number), char (character), string (a string of characters), etc.
  - Its value CANNOT change after it is initialized or assigned for the first time

```
final double  PI = 3.14;
final  int  myconstant = 34;
final  float  x = 4.76;
```

# Modifying Variables

# Modifying Variables

- Variables can be changed by:
  1. Assigning to them
     - Using assignment statement: stores a new value in a variable, replacing the previously stored value
  2. Incrementing or decrementing them
     - Using increment/decrement operator
  3. Inputting into them
     - Input from mouse or keyboard

# Assignment Statement

- The = in an assignment does ***not*** mean the left hand side is equal to the right hand side as it does in math.
- **=** is an instruction to do something:

  ***copy*** the value of the expression on the right
  ***into*** the variable on the left.

```
int counter = 11; // set counter to 11
counter = counter + 1; // increment
```

# Assignment Statement

```
int counter = 11; // set counter to 11
counter = counter + 1; // increment
```

1. Look up what is currently in counter (11)
2. Add 1 to that value (12)
3. Copy the result of the addition expression into the variable on the left, changing counter

```
System.out.println (counter);
```

12 is shown

# 2. Increment & Decrement

- There are two unary increment / decrement operators:
  - `++` (called plus plus)
  - `--` (called minus minus)

- **Prefix operator**
  - Placed before a variable
- **Postfix operator**
  - Placed after a variable

- They can be prefix, or postfix operators;

Postfix increment: `x++;`

Prefix increment: `++x;`

Postfix decrement: `x--;`

Prefix decrement: `--x;`

**Decrements first**
Then uses x

**Uses x first**
Then decrements

# Using ++ and --

```
int x = 5, y = 2;
int z;

z = x--;   // first uses x then decrements x
System.out.println(z);  // prints 5
System.out.println(x);  // prints 4
```

# Using ++ and --

```
int x = 5, y = 2;
int z;

z = x++;    // first uses x then increments x
System.out.println(z);  // prints 5
System.out.println(x);  // prints 6
```

# Using ++ and --

```
int x = 5, y = 2;
int z;

z = ++y;    // first increments y then uses y
System.out.println(z);  // prints 3
System.out.println(y);  // prints 3
```

# Using ++ and --

```
int x = 5, y = 2;
int z;

z = --y;    // first decrements y then uses y
System.out.println(z);  // prints 1
System.out.println(y);  // prints 1
```