Al-Mustaqbal University
College of Sciences
Intelligent Medical System Department

جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم
قسم الانظمة الطبية الذكية

Lecture: (5)

Basic Computation Part II

Subject: Computer Programming (I)
Level: First
Lecturer: Dr. Maytham N. Meqdad

• Java Identifiers

All Java variables must be identified with unique names.
These unique names are called identifiers.
Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

The general rules for naming variables are:

- Names can contain letters, digits, underscores, and dollar signs
- Names must begin with a letter
- Names should start with a lowercase letter and it cannot contain whitespace
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like Java keywords, such as `int` or `boolean`) cannot be used as names

• Java Variables

Variables are containers for storing data values.

In Java, there are different **types** of variables, for example:

- String - stores text, such as "Hello". String values are surrounded by double quotes
- int - stores integers (whole numbers), without decimals, such as 123 or -123
- float - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- boolean - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

```
type variableName = value;
```

Where *type* is one of Java's types (such as `int` or `String`), and *variableName* is the name of the variable (such as **x** or **name**). The **equal sign** is used to assign values to the variable.

To create a variable that should store text, at the following example:

Example

Create a variable called **name** of type `String` and assign it the value "**John**":

```
String name = "John";
System.out.println(name);
```

To create a variable that should store a number, at the following example:

Example

Create a variable called **myNum** of type `int` and assign it the value **15**:

```
int myNum = 15;
System.out.println(myNum);
```

You can also declare a variable without assigning the value, and assign the value later:

```
int myNum;
myNum = 15;
System.out.println(myNum);
```

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example

Change the value of `myNum` from `15` to `20`:

```
int myNum = 15;
myNum = 20;  // myNum is now 20
System.out.println(myNum);
```

Final Variables

If you don't want others (or yourself) to overwrite existing values, use the `final` keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

Example

```
final int myNum = 15;
myNum = 20;  // will generate an error: cannot assign a value to a final
variable
```

Other Types

A demonstration of how to declare variables of other types:

Example

```
int myNum = 5;
float myFloatNum = 5.99f;
char myLetter = 'D';
boolean myBool = true;
String myText = "Hello";
```

• Java Naming Conventions

A naming convention is a rule to follow as you decide what to name your identifiers (e.g. class, package, variable, method, etc.).

Why Use Naming Conventions?

Different Java programmers can have different styles and approaches to the way they program. By using standard Java naming conventions they make their code easier to read for themselves and for other programmers. Readability of Java code is important because it means less time is spent trying to figure out what the code does, leaving more time to fix or modify it.

To illustrate the point it's worth mentioning that most software companies will have a document that outlines the naming conventions they want their programmers to follow. A new programmer who becomes familiar with those rules will be able to understand code written by a programmer who might have left the company many years beforehand.

Picking a Name for Your Identifier

When choosing a name for an identifier, make sure it's meaningful. For instance, if your program deals with customer accounts then choose names that make sense for dealing with customers and their accounts (e.g., customerName, accountDetails). Don't worry about the length of the name. A longer name that sums up the identifier perfectly is preferable to a shorter name that might be quick to type but ambiguous.

A Few Words About Cases

Using the right letter case is the key to following a naming convention:

- **Lowercase** is where all the letters in a word are written without any capitalization (e.g., while, if, mypackage).
- **Uppercase** is where all the letters in a word are written in capitals. When there are more than two words in the name use underscores to separate them (e.g., MAX_HOURS, FIRST_DAY_OF_WEEK).
- **CamelCase** (also known as Upper CamelCase) is where each new word begins with a capital letter (e.g., CamelCase, CustomerAccount, PlayingCard).
- **Mixed case** (also known as Lower CamelCase) is the same as CamelCase except the first letter of the name is in lowercase (e.g., hasChildren, customerFirstName, customerLastName).

Standard Java Naming Conventions

The below list outlines the standard Java naming conventions for each identifier type:

- **Packages:** Names should be in lowercase. With small projects that only have a few packages it's okay to just give them simple (but meaningful!) names:

```
package pokeranalyzer package mycalculator
```

  In software companies and large projects where the packages might be imported into other classes, the names will normally be subdivided. Typically this will start with the company domain before being split into layers or features:

```
package com.mycompany.utilities package
org.bobscompany.application.userinterface
```

- **Classes:** Names should be in CamelCase. Try to use nouns because a class is normally representing something in the real world:

```
class Customer class Account
```

- **Interfaces:** Names should be in CamelCase. They tend to have a name that describes an operation that a class can do:

```
interface Comparable interface Enumerable
```

  Note that some programmers like to distinguish interfaces by beginning the name with an "I":

```
interface IComparable interface IEnumerable
```

- **Methods:** Names should be in mixed case. Use verbs to describe what the method does:

```
void calculateTax() string getSurname()
```

- **Variables:** Names should be in mixed case. The names should represent what the value of the variable represents:

```
string firstName int orderNumber
```

  Only use very short names when the variables are short-lived, such as in for loops:

```
for (int i=0; i<20;i++) {   //i only lives in here }
```

- **Constants:** Names should be in uppercase.

```
static final int DEFAULT_WIDTH static final int MAX_HEIGHT
```

• Constant Variables

A constant is a variable whose value cannot change once it has been assigned. Java doesn't have built-in support for constants, but the variable modifiers *static* and *final* can be used to effectively create one.

Constants can make your program more easily read and understood by others. In addition, a constant is cached by the JVM as well as your application, so using a constant can improve performance.

Static Modifier

This allows a variable to be used without first creating an instance of the class; a static class member is associated with the class itself, rather than an object. All class instances share the same copy of the variable.

This means that another application or main() can easily use it.

For example, class myClass contains a static variable days_in_week:

```
public class myClass {
   static int days_in_week = 7;
}
```

Because this variable is static, it can be used elsewhere without explicitly creating a myClass object:

```
public class myOtherClass {
   static void main(String[] args) {
       System.out.println(myClass.days_in_week);
   }
 }
```