Object-Oriented Programming with Java
Methods

Quan Thanh Tho, Ph.D.
CSE – HCMUT
qttho@cse.hcmut.edu.vn

Methods

- Hello World!
- Java program compilation
- Introducing Methods
- Declaring Methods
- Calling Methods
- Passing Parameters by value
- Overloading Methods

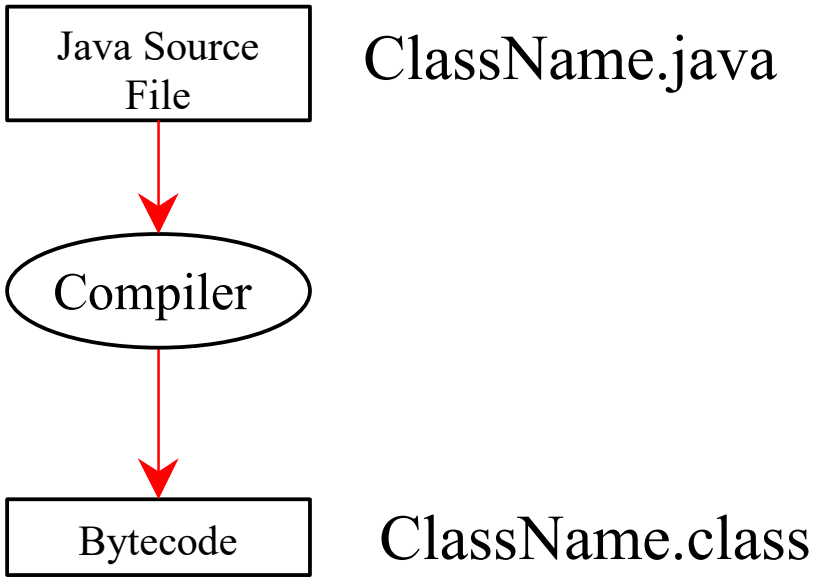
A Simple Application

Example 1

```
//This application program prints "Hello World!"
public class HelloWorld
{
   int n = 10;
   public static void main(String[] args)
   {
      System.out.println("Hello  World " +
                                n + " times!");
   }
}
```

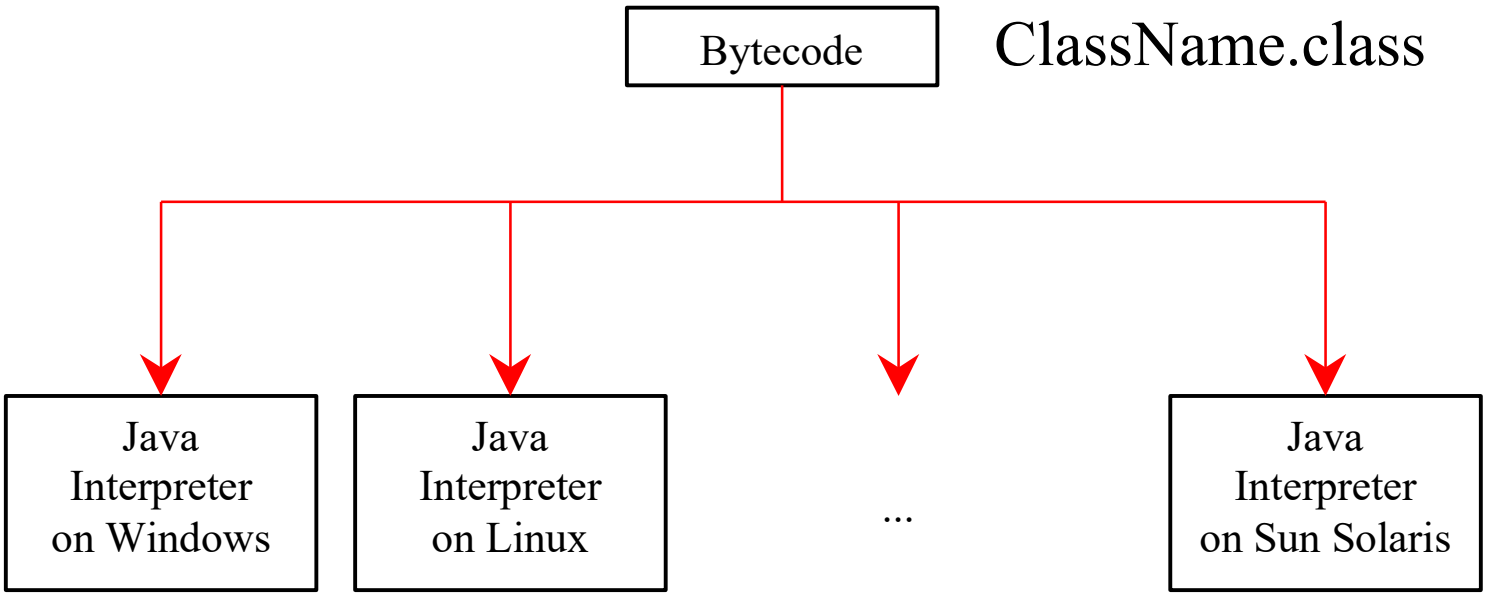
Compiling Java Programs

- Command line on cs
  - `javac ClassName.java`



Executing Applications

- Command line on cs
  - `java ClassName`



Example 1

- `javac HelloWorld.java`
- `java HelloWorld`
- `Hello World 10 times!`

Compiling and execution

- Compilation:

```
javac HelloWorld.java
```

Result of compilation of Java file `HelloWorld.java` is file `HelloWorld.class` with bytecode

- Execution

```
java HelloWorld
```

Result is “HelloWorld 10 times!” to standard output

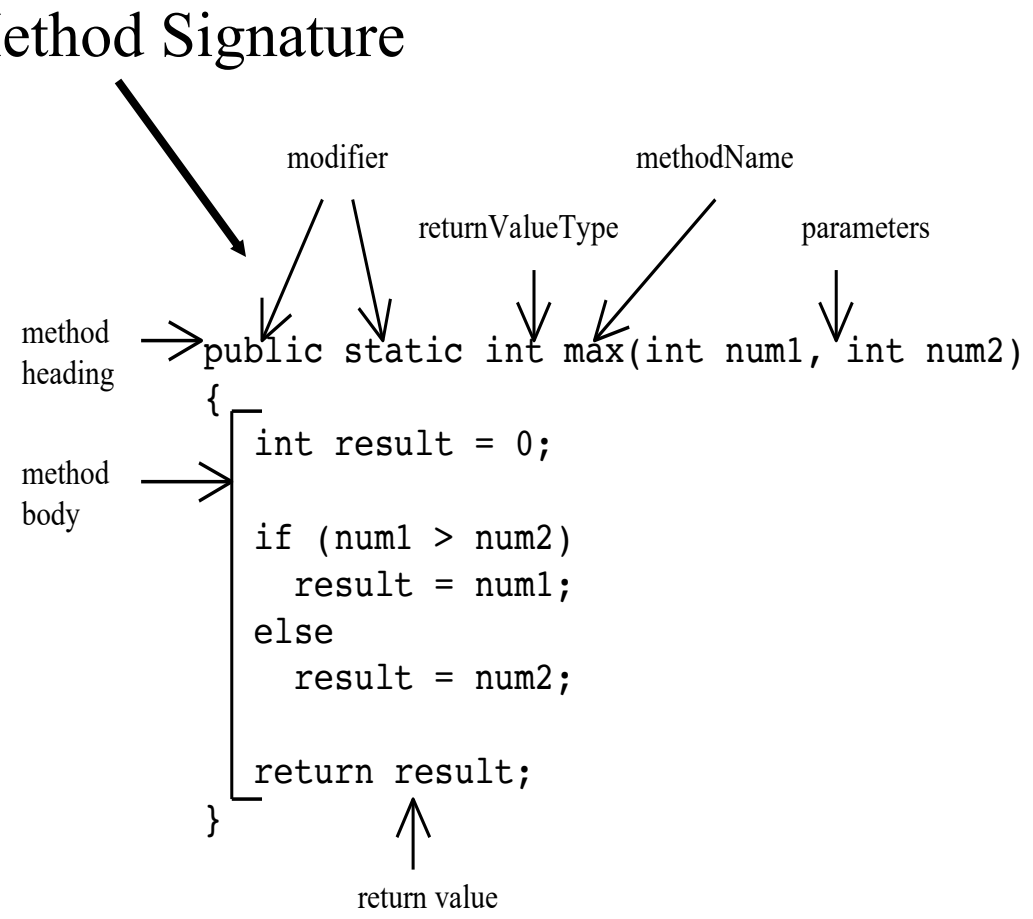
Simple skeleton of Java application

File: `ProgramName.java`

```
public class ProgramName
{
    // Define program variables here.
    double d;
    ...
    // Define program methods here.
    int method1()
    { // Do something
    }
    . . .
    //Define the main method here.
    public static main(String args[])
    {
        // Main method body
    }//end of the main method.
} //End of class ProgramName
```

Introducing Methods

A method is a collection of statements that are grouped together to perform an operation.



Method signature

The combined name and parameter list for each method in a class must be unique. The uniqueness of a parameter list takes the order of the parameters into account.

So `int myMethod (double q, int n)` is unique from
`int myMethod (double q, double x)` and

`int myMethod (int k, double y).`

Declaring methods

```
[modifiers] return_type method_name (parameter_list)
{
    [statement_list]
}
```

Everything within square brackets [] is optional.

The minimal method declaration includes:

- *Return Type*: The return type is either a valid Java type (primitive or class) or `void` if no value is returned. If the method declares a return type, every exit path out of the method must have a return statement.
- *Method Name*: The method name must be a valid Java identifier.
- *Parameter List*: The parentheses following the method name contain zero or more type/identifier pairs that make up the parameter list. Each parameter is separated by a comma. Also, there can be zero parameters.

Declaring Methods

```
int max(int num1, int num2)
{
   int x;
   if (num1 > num2)
      x = num1;
   else
      x = num2;
   return x;
}
```

Calling Methods

```
public class TestMax
{
  /**A method for finding a max between two numbers*/
  int max(int num1, int num2)
  {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }

  public static void main(String[] args)
  {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i +
      " and " + j + " is " + k);
  }
}
```

Passing parameters by value

- When a primitive value is passed into a method, a copy of the primitive is made. The copy is what is actually manipulated in the method.
- So, the value of the copy can be changed within the method, but the original value remains unchanged.

Passing parameters by value

```
int myMethod(int a, int n)
{
  int     S = 0;

  for (int i=0; i<=n; i++)
  {
     S += a;
     a++;
  }
  return S;
}
-------------------------

a = 10;
System.output.printkn("a="+a); // a=10
int b = myMethod(a,5);
System.output.println("a="+a); // a=?
```

Passing parameters by value

```
int myMethod(int a, int n)
{
    int     S = 0;

    for (int i=0; i<=n; i++)
   {
       S += a;
       a++;
   }
    return S;
}
-----------------------


a = 10;
System.output.println("a="+a); // a=10
int b = myMethod(a,5);
System.output.println("a="+a); // a=10
```

Polymorphism: Overloading Methods

- The practice of defining more than one method *in a class* with same name is called *method overloading*.
- Java resolves overloaded method names using the *types* of the argument expressions.
- When the Java compiler encounters a method invocation involving an overloaded method, it selects the ``best'' (most specific) match from among the alternatives.
- If no best method exists, the program is ill-formed and will be rejected by the Java compiler.

Overloading Methods

```
int max(int num1, int num2)
{
  if (num1 > num2)
    return num1;
  else
    return num2;
}
double max(double num1, double num2)
{
  if (num1 > num2)
    return num1;
  else
    return num2;
}
```