Java File Naming Rules - Complete Guide

(Formatted for Word Document)

Key Rules Summary

✅ **Public Class** → File **must** match class name (e.g., `public class Car` → `Car.java`)
✅ **Non-Public Class** → Any filename allowed (but prefer `main()` class name)
❌ **Multiple Public Classes** → Not allowed (Compiler error)

---

1. Public Class Examples

✓ Valid Case

Filename: `Employee.java`

```
java

Copy

public class Employee {  // Public class → filename MUST be Employee.java
    public static void main(String[] args) {
        System.out.println("Valid Example");
    }
}
```

Output:

```
Copy

> javac Employee.java
> java Employee
Valid Example
```

❌ Invalid Case

Filename: `Main.java` *(Wrong name for public class)*

```
java

Copy

public class Employee { }  // Error: Must be Employee.java
```

Compiler Error:

```
Copy
Main.java:1: error: class Employee is public, should be declared in Employee.java
```

---

2. Non-Public Class Examples

✓ Arbitrary Filename Allowed

Filename: abc.java

```
java
Copy
class Test {  // Non-public → any filename works
    public static void main(String[] args) {
        System.out.println("Works!");
    }
}
```

Run with:

```
Copy
> javac abc.java
> java Test
Works!
```

---

3. Multiple Public Classes (Error)

```
java
Copy
public class A { }  // Needs A.java
public class B { }  // Needs B.java → ERROR in single file
```

Compiler Output:

```
Copy
A.java:2: error: class B is public, should be in B.java
```

Quick Reference Table

Case	Valid Filename	Example
Public Class	ClassName.java	Student.java
Non-Public Class	Any name	Main.java, a.java
Multiple Public Classes	❌ Not Allowed	Compiler Error

Best Practices

✓ **1 class per file** (Easier maintenance)
✓ **Match filename to public class**
✓ **Use main() class name for non-public files**

Java File Compilation Notes

What is Compilation in Java?

Compilation in Java is the process of translating high-level Java code (written by developers) into low-level bytecode (.class files) that can be executed by the Java Virtual Machine (JVM). The Java compiler (javac) performs this task.

Why is Compilation Required?

1. **Translation**: Converts high-level code into low-level bytecode representations.
2. **Error Checking**: Identifies and reports mistakes (syntax errors, etc.) in the Java program.

Command for Compilation

Java provides the javac command to compile Java files:

text
CollapseWrapCopy

```
javac FileName.java
```

Example:

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>javac Test.java
```

Requirements and Process for Compilation

Prerequisites

1. **Java Development Kit (JDK)**: Must be installed (e.g., jdk1.8.0).
2. **Path Environment Variable**: Must be set to the JDK's bin directory where javac resides.

   text

   CollapseWrapCopy

   ```
   set path=C:\Java\jdk1.8.0\bin;
   ```

   - If not set, the OS returns:

     text

     CollapseWrapCopy

     ```
     'javac' is not recognized as an internal or external command,
     operable program or batch file.
     ```

Compilation Process

When you run javac FileName.java, the following happens:

1. **OS Action**:
   - Searches for javac in its internal command list and the path variable.
   - If found, executes the Java compiler; otherwise, shows an error.
2. **Compiler Actions**: a. Takes the specified .java file from the command prompt. b. Searches for the file in the current directory. c. If the file is not found:

   text

   CollapseWrapCopy

   ```
   javac: file not found: Test.java
   ```

   d. If found, compiles the file from start to end. e. If errors exist, displays error messages. f. If no errors, generates .class files.

Output of Compilation: .class Files

- **Location**: By default, .class files are generated in the same directory as the .java file.
- **Number of .class Files**: Depends on the number of classes, abstract classes, interfaces, enums, and inner classes in the file, **not** the number of .java files.

Example

Code (Test.java):

java
CollapseWrapCopy

```java
enum E {}
interface I {}
abstract class A {}
class B {
    class C {}
}
class Test {
    public static void main(String[] args) {
        System.out.println("Welcome To Java programming.....");
    }
}
```

Command:

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>javac Test.java
```

Output (Directory Listing):

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>dir
 Volume in drive C has no label.
 Volume Serial Number is 3E3B-4439
 Directory of C:\apps\COREJAVA-APPS
07/30/2024  09:13 PM    <DIR>          .
07/21/2024  10:25 AM    <DIR>          ..
07/30/2024  09:13 PM               179 A.class
07/30/2024  09:13 PM               270 B$C.class
07/30/2024  09:13 PM               223 B.class
07/30/2024  09:13 PM               611 E.class
07/30/2024  09:13 PM                86 I.class
07/30/2024  09:13 PM               434 Test.class
07/30/2024  09:12 PM               202 Test.java
```

- **Generated Files**: A.class, B$C.class, B.class, E.class, I.class, Test.class.

Customizing .class File Location with -d Option

To specify a different output directory for .class files, use the -d option:

text
CollapseWrapCopy

```
javac -d targetLocation FileName.java
```

Example

Code (Test.java):

java
CollapseWrapCopy

```java
enum E {}
interface I {}
abstract class A {}
class B {
    class C {}
}
class Test {
    public static void main(String[] args) {
        System.out.println("Welcome To Java programming.....");
    }
}
```

Command:

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>javac -d C:\target Test.java
```

Output (Directory Listing):

text
CollapseWrapCopy

```
C:\target>dir
 Volume in drive C has no label.
 Volume Serial Number is 3E3B-4439
 Directory of C:\target
```

```
07/30/2024  09:25 PM    <DIR>          .
07/30/2024  09:25 PM               179 A.class
07/30/2024  09:25 PM               270 B$C.class
07/30/2024  09:25 PM               223 B.class
07/30/2024  09:25 PM               611 E.class
07/30/2024  09:25 PM                86 I.class
07/30/2024  09:25 PM               434 Test.class
```

Compilation with Package Declaration

When a Java file includes a package statement, use -d to generate the folder structure corresponding to the package name.

Example

Code (Test.java):

java
CollapseWrapCopy

```java
package com.durgasoft.core;
enum E {}
interface I {}
abstract class A {}
class B {
    class C {}
}
class Test {
    public static void main(String[] args) {
        System.out.println("Welcome To Java programming.....");
    }
}
```

Command:

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>javac -d C:\target Test.java
```

Output (Directory Listing):

text
CollapseWrapCopy

```
C:\target\com\durgasoft\core>dir
 Volume in drive C has no label.
 Volume Serial Number is 3E3B-4439
 Directory of C:\target\com\durgasoft\core
07/30/2024  09:33 PM    <DIR>          .
07/30/2024  09:33 PM    <DIR>          ..
07/30/2024  09:33 PM               198 A.class
07/30/2024  09:33 PM               346 B$C.class
07/30/2024  09:33 PM               261 B.class
07/30/2024  09:33 PM               706 E.class
07/30/2024  09:33 PM               105 I.class
07/30/2024  09:33 PM               453 Test.class
```

Compiling Multiple Java Files with a Single Command

Yes, it's possible to compile multiple .java files using a single javac command. Below are the cases:

Case 1: Listing Multiple Files

Provide all file names with a space separator.

text
CollapseWrapCopy

```
javac file1.java file2.java file3.java
```

Example:

- **A.java**:

java
CollapseWrapCopy

```
public class A {}
```

- **B.java**:

java
CollapseWrapCopy

```
public class B {}
```

- **C.java**:

java
CollapseWrapCopy

```
public class C {}
```

- **Test.java**:

java
CollapseWrapCopy

```
public class Test {}
```

Command:

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>javac A.java B.java C.java Test.java
```

Output:

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>dir
 Volume in drive C has no label.
 Volume Serial Number is 3E3B-4439
 Directory of C:\apps\COREJAVA-APPS
08/01/2024  08:36 AM    <DIR>          .
07/21/2024  10:25 AM    <DIR>          ..
08/01/2024  08:36 AM               176 A.class
08/01/2024  08:33 AM                27 A.java
08/01/2024  08:36 AM               176 B.class
08/01/2024  08:33 AM                27 B.java
08/01/2024  08:36 AM               176 C.class
08/01/2024  08:34 AM                27 C.java
08/01/2024  08:36 AM               434 Test.class
08/01/2024  08:33 AM               123 Test.java
```

Case 2: Compiling Dependent Files

If files depend on each other, compiling the base file compiles all dependent files.

Example:

- **Test.java**:

java
CollapseWrapCopy

```
class Test {
    A a = new A();
    public static void main(String[] args) {
        System.out.println("Welcome To Java programming.....");
    }
}
```

- **A.java**:

java
CollapseWrapCopy

```
public class A {
    B b = new B();
}
```

- **B.java**:

java
CollapseWrapCopy

```
public class B {
    C c = new C();
}
```

- **C.java**:

java
CollapseWrapCopy

```
public class C {}
```

Command:

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>javac Test.java
```

Output:

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>dir
 Volume in drive C has no label.
 Volume Serial Number is 3E3B-4439
 Directory of C:\apps\COREJAVA-APPS
08/01/2024  08:43 AM    <DIR>          .
```

```
07/21/2024  10:25 AM    <DIR>          ..
08/01/2024  08:43 AM               231 A.class
08/01/2024  08:41 AM                41 A.java
08/01/2024  08:43 AM               231 B.class
08/01/2024  08:41 AM                41 B.java
08/01/2024  08:43 AM               176 C.class
08/01/2024  08:41 AM                27 C.java
08/01/2024  08:43 AM               489 Test.class
08/01/2024  08:42 AM               140 Test.java
```

Case 3: Using Wildcards (*)

Use * notation to compile multiple files based on patterns.

Command	Description
javac *.java	Compiles all .java files in the directory.
javac *Address.java	Compiles files ending with Address.
javac Employee*.java	Compiles files starting with Employee.
javac *Account*.java	Compiles files containing Account.

Example:

- **CustomerAddress.java**:

java
CollapseWrapCopy

```
public class CustomerAddress {}
```

- **CustomerMails.java**:

java
CollapseWrapCopy

```
public class CustomerMails {}
```

- **EmployeeAccountDetails.java**:

java
CollapseWrapCopy

```
public class EmployeeAccountDetails {}
```

- **EmployeeAddress.java**:

java
CollapseWrapCopy

```
public class EmployeeAddress {}
```

- **EmployeeMails.java**:

java
CollapseWrapCopy

```
public class EmployeeMails {}
```

- **StudentAccountDetails.java**:

java
CollapseWrapCopy

```
public class StudentAccountDetails {}
```

- **StudentAddress.java**:

java
CollapseWrapCopy

```
public class StudentAddress {}
```

- **StudentMails.java**:

java
CollapseWrapCopy

```
public class StudentMails {}
```

Commands:

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>javac *.java
C:\apps\COREJAVA-APPS>javac Employee*.java
C:\apps\COREJAVA-APPS>javac *Address.java
C:\apps\COREJAVA-APPS>javac *Account*.java
```

Output:

text
CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>dir
 Volume in drive C has no label.
 Volume Serial Number is 3E3B-4439
 Directory of C:\apps\COREJAVA-APPS
08/01/2024  09:02 AM    <DIR>          .
07/21/2024  10:25 AM    <DIR>          ..
```

```
08/01/2024  09:02 AM               204 CustomerAddress.class
08/01/2024  08:50 AM                41 CustomerAddress.java
08/01/2024  09:02 AM               200 CustomerMails.class
08/01/2024  08:50 AM                39 CustomerMails.java
08/01/2024  09:03 AM               218 EmployeeAccountDetails.class
08/01/2024  09:00 AM                48 EmployeeAccountDetails.java
08/01/2024  09:02 AM               204 EmployeeAddress.class
08/01/2024  08:49 AM                41 EmployeeAddress.java
08/01/2024  09:02 AM               200 EmployeeMails.class
08/01/2024  08:50 AM                39 EmployeeMails.java
08/01/2024  09:03 AM               216 StudentAccountDetails.class
08/01/2024  08:56 AM                47 StudentAccountDetails.java
08/01/2024  09:02 AM               202 StudentAddress.class
08/01/2024  08:50 AM                40 StudentAddress.java
08/01/2024  09:02 AM               198 StudentMails.class
08/01/2024  08:50 AM                38 StudentMails.java
```

Key Points

1. Set the path variable to use javac.
2. .class files are generated based on the number of classes, not files.
3. Use -d to specify output directory or handle packages.
4. Compile multiple files using space-separated names, dependencies, or wildcards.

Notes on Executing Java Applications

Overview of Java Execution

To execute a Java application, Java provides the java command, which runs the compiled .class file containing the main method. The Java Virtual Machine (JVM) is responsible for executing the bytecode.

Command for Execution

text

CollapseWrapCopy

```
java MainClassName
```

- **Note**: This command must be run in the directory where the main class's .class file exists.

Example:

text

CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>java Test
```

Execution Process

When the java command is executed, the following steps occur:

1. Operating System (OS) Actions

- The OS takes the java command from the command prompt.
- It searches for the java command in:
  - Its internal command list.
  - Locations specified by the path environment variable (e.g., C:\Java\jdk1.8.0\bin).
- If the java command is found, the OS executes it, activating the JVM.
- If not found, the OS displays:

  text

  CollapseWrapCopy

  ```
  'java' is not recognized as an internal, external command, an
  operable program and batch file.
  ```

Diagram Reference (C and E):

- **C**: Shows the JDK structure (C:\Java\jdk1.8.0\bin) containing java, javac, etc.
- **E**: Highlights the path variable setup:

  text

  CollapseWrapCopy

  ```
  path=.;-;C:\Java\jdk1.8.0\bin;
  ```

Example:

text

CollapseWrapCopy

```
D:\java830>javac Test.java
```

```
'javac' is not recognized as an internal, external command, an operable
program and batch file.



D:\java830>set path=C:\Java\jdk1.8.0\bin;

D:\java830>javac Test.java
```

---

2. JVM Actions

Once the JVM is activated, it performs the following steps:

Step 1: Take the Main Class Name

- The JVM extracts the main class name (Test in java Test) from the command prompt.

Step 2: Search for the .class File

- The JVM searches for the main class's .class file (Test.class) in:
  - The current directory.
  - Java's predefined library.
  - Locations specified by the classpath environment variable (e.g., .;-;E:\abc).
- If the .class file is not found, the JVM throws an error:
  - **Java 6**:

    text

    CollapseWrapCopy

    ```
    java.lang.NoClassDefFoundError: Test
    ```

  - **Java 7**:

    text

    CollapseWrapCopy

    ```
    Error: Could not find or load main class Test
    ```

Diagram Reference (D and E):

- **D**: Shows the Test.java file and Test.class file in the java830 directory.
- **E**: Shows the classpath setup:

  text

CollapseWrapCopy

```
classpath=.;-;E:\abc;
```

Example:

text

CollapseWrapCopy

```
D:\java830>java Test

java.lang.NoClassDefFoundError: Test



D:\java830>set classpath=E:\abc;

D:\java830>java Test

Welcome To Java Programming...
```

Step 3: Set the classpath if Needed

- If the .class file is in a different location, set the classpath to point to that location:

  text

  CollapseWrapCopy

  ```
  set classpath=TargetLocation
  ```

Example:

- **Code** (Test.java):

  java

  CollapseWrapCopy

  ```
  public class Test {

      public static void main(String[] args) {

          System.out.println("Welcome To Java Programming...");

      }

  }
  ```

- **Commands**:

  text

  CollapseWrapCopy

```
C:\apps\COREJAVA-APPS>javac -d C:\target Test.java

C:\apps\COREJAVA-APPS>set classpath=C:\target;

C:\apps\COREJAVA-APPS>java Test
```

- **Output**:

  text

  CollapseWrapCopy

```
Welcome To Java Programming...
```

Step 4: Load the .class File

- If the .class file is found, the JVM loads its bytecode into memory.

Step 5: Search for the main() Method

- The JVM looks for the public static void main(String[] args) method in the loaded class.
- If the main() method is not found, the JVM throws an error:
  - **Java 6**:

    text

    CollapseWrapCopy

```
java.lang.NoSuchMethodError: main
```

  - **Java 7**:

    text

    CollapseWrapCopy

```
Error: Main method not found in class Test, please define the
main method as: public static void main(String[] args)
```

Step 6: Create the Main Thread

- If the main() method exists, the JVM creates a thread called the **Main Thread** to execute it.

Diagram Reference (D):

- **D**: Shows the Main Thread executing the main() method in Test.class.

Step 7: Execute the main() Method

- The Main Thread starts executing the main() method from its starting point to its ending point.
- In the example, it prints:

  text

  CollapseWrapCopy

```
Welcome To Java Programming...
```

Step 8: Main Thread Reaches Dead State

- When the Main Thread reaches the end of the main() method, it enters the **Dead State**.

Diagram Reference (D):

- **D**: Marks the Dead State of the Main Thread after execution.

Step 9: JVM Shutdown

- Once the Main Thread is in the Dead State, the JVM:
  - Stops all internal processes related to the Java program.
  - Enters **Shutdown Mode**.

Diagram Explanation

The diagram illustrates the Java execution process:

- **C (JDK Structure)**:
  - Shows the C:\Java\jdk1.8.0 directory with bin containing javac, java, etc.
  - The OS (via CMD) interacts with this directory to find the java command.
- **D (Execution Flow)**:
  - Depicts the java830 directory with Test.java and Test.class.
  - Shows the Main Thread executing the main() method and reaching the Dead State.
  - Indicates that Test.class is not found in the java830 directory initially, leading to errors.
- **E (Environment Variables and Commands)**:

- Lists the path and classpath variables:

  text

  CollapseWrapCopy

```
path=.;-;C:\Java\jdk1.8.0\bin;

classpath=.;-;E:\abc;
```

- Shows the sequence of commands and errors:

  text

  CollapseWrapCopy

```
D:\java830>javac Test.java

'javac' is not recognized as an internal, external command, an
operable program and batch file.



D:\java830>set path=C:\Java\jdk1.8.0\bin;

D:\java830>javac Test.java



D:\java830>java Test

java.lang.NoClassDefFoundError: Test



D:\java830>set classpath=E:\abc;

D:\java830>java Test

Welcome To Java Programming...
```

Key Points

- The java command must be run where the .class file exists or set the classpath.
- The path variable must include the JDK's bin directory to locate java and javac.
- The JVM loads the .class file, executes the main() method via the Main Thread, and shuts down after execution.
- Errors occur if:
    - The java command is not found (path not set).
    - The .class file is not found (classpath not set).
    - The main() method is missing or incorrectly defined.

Table: Common Errors and Solutions

Error	Cause	Solution
'java' is not recognized...	java not in path	set path=C:\Java\jdk1.8.0\bin;
java.lang.NoClassDefFoundError: Test	.class file not found	set classpath=TargetLocation;
Error: Could not find or load main class	.class file not found (Java 7)	set classpath=TargetLocation;
java.lang.NoSuchMethodError: main	main() method missing (Java 6)	Define public static void main(String[] args)
Error: Main method not found in class...	main() method missing (Java 7)	Define public static void main(String[] args)