NYIT

**Fall 2021**

Homework No: 03

Title: Java Object, Class, Packages

**Name:** Patade, Yash Arun

**Class ID#:**  11

**School ID#:** 1284979

**Course:** Java Networking

**Course ID:** CSCI 725

**Date:** 01/10/2021

Assignment Contents

[1) Java Class as a Basic Java Code Unit. 6](#_Toc85588493)

[2) Packages. 7](#_Toc85588494)

[3) Package Syntax – Name Scope. 9](#_Toc85588495)

[ Java provides a syntax for package names. 9](#_Toc85588496)

[4) Java Package as Code Organizational Unit. 9](#_Toc85588497)

[5) Packages and Name Space Collision. 11](#_Toc85588498)

[6) Question: Packages. 13](#_Toc85588499)

[ What is the meaning of name space collision? 13](#_Toc85588500)

[7) Adding to the Package. 13](#_Toc85588501)

[ At OOP/compile-time, to have class under development belong to a particular package use Java package operator. 13](#_Toc85588502)

[8) Multiple Classes in the Package ***first***. 13](#_Toc85588503)

[9) Two classes in Different Packages. 14](#_Toc85588504)

[10) Making a Class Belong to a Package. 14](#_Toc85588505)

[ Include a proper package statement as first line in source file. 14](#_Toc85588506)

[ Make class ***B*** belong to ***packA*** 15](#_Toc85588507)

[ Make class A belong to package pC: 15](#_Toc85588508)

[11) Importing the package. 16](#_Toc85588509)

[ Import statement allows importing or linking of a package. 16](#_Toc85588510)

[12) Importing or Linking ***packX.packA.pA;*** 16](#_Toc85588511)

[13) Default Package. 17](#_Toc85588512)

[14) Default Package Convenience. 18](#_Toc85588513)

[ The use of default packages (current directory) is convenient. 18](#_Toc85588514)

[15) Java and a Global Namespace. 18](#_Toc85588515)

[• The designers of Java have proposed an Internet-wide, i.e., Web-wide unique package naming scheme. 18](#_Toc85588516)

[16) Putting Classes in Packages. 19](#_Toc85588517)

[• To put a class into a package, one uses the "package" statement. 19](#_Toc85588518)

[17) Java and a Global Namespace. 20](#_Toc85588519)

[18) Using ***import*** to Simplify Code. 21](#_Toc85588520)

[ Unfortunately, the proposed naming scheme creates very long names for classes. 21](#_Toc85588521)

[19) Package Inclusion Simplifies Code Text. 22](#_Toc85588522)

[20) Import statement. 23](#_Toc85588523)

[• Options for the import statement: 23](#_Toc85588524)

[21) Importing the Package at Run Time. 23](#_Toc85588525)

[• At run time JRE looks at two places for user created packages: 24](#_Toc85588526)

[22) CLASSPATH Environmental Variables. 24](#_Toc85588527)

[23) Question: CLASSPATH Environment Variables. 24](#_Toc85588528)

[ What should be true in order for your program to find package ***mypack***? 24](#_Toc85588529)

[Answer: 24](#_Toc85588530)

[24) Resolving Packages at Runtime. 25](#_Toc85588531)

[• Java compiler compiles .***java*** files to .***class*** files. 25](#_Toc85588532)

[25) Is CLASSPATH important at Compile Time? 25](#_Toc85588533)

[• CLASSPATH is also used by the compiler to resolve association issues at compile time: 25](#_Toc85588534)

[• However, what if the class has not yet been compiled and no .***class*** file exists? 25](#_Toc85588535)

[26) Compiling Classes with Packages. 26](#_Toc85588536)

[ Packages provide a logical namespace for classes 26](#_Toc85588537)

[27) .***jar*** files as Java Packages. 27](#_Toc85588538)

[28) .***jar*** File Containing the Main Class. 27](#_Toc85588539)

[ JVM class loader checks specified packages by a programmer and using jar –x extracts needed classes. 27](#_Toc85588540)

[29) Defining Abstraction. 28](#_Toc85588541)

[ Abstraction is the process of recognizing and extraction common features from a set of specific example-objects/samples. 28](#_Toc85588542)

[30) Different types of abstraction. 28](#_Toc85588543)

[ Data Abstraction 29](#_Toc85588544)

[ Functional Abstraction 29](#_Toc85588545)

[ Object Abstraction 29](#_Toc85588546)

[31) Defining a Java Class. 29](#_Toc85588547)

[ A Java Class denotes a category of objects and acts as a blueprint for creating such objects. 29](#_Toc85588548)

[Class Modifiers. 29](#_Toc85588549)

[ Class modifiers change the way a class can be used. 29](#_Toc85588550)

[Access Modifiers. 30](#_Toc85588551)

[ Member modifiers change the way class members can be used. 30](#_Toc85588552)

[Member Modifiers. 31](#_Toc85588553)

[ Member modifiers change the way class members can be used. 31](#_Toc85588554)

[Accessibility Scope. 32](#_Toc85588555)

[ Accessibility scope defines the boundary of access to a class and its members 32](#_Toc85588556)

[Defining Encapsulation. 32](#_Toc85588557)

[ Encapsulation is the process of hiding an object’s implementation from another object, while presenting only the interfaces that should be visible. 32](#_Toc85588558)

[Encapsulating a Class. 32](#_Toc85588559)

[ Members of a class must always be declared with the minimum level of visibility. 33](#_Toc85588560)

[Setters and Getters. 33](#_Toc85588561)

[ Setters and Getters allow controlled access to class data. 33](#_Toc85588562)

[Basic Archives Procedures Overview. 34](#_Toc85588563)

[The ***jar*** Command. 34](#_Toc85588564)

[ The ***jar*** command is similar to the tar command, but compresses the resulting file in the same step. 34](#_Toc85588565)

[ The syntax for the ***jar*** tool is almost identical to the syntax for the tar command. 35](#_Toc85588566)

[The ***jar*** Command Option. 35](#_Toc85588567)

[Chapter 5 Homework. 36](#_Toc85588568)

[1) Decimal to Hex. 36](#_Toc85588569)

[2) Future Tuition. 37](#_Toc85588570)

[3) Greatest Common Divisor. 38](#_Toc85588571)

[4) Guess Number. 39](#_Toc85588572)

[5) Guess Number One Time. 41](#_Toc85588573)

[6) Multiplication Table. 43](#_Toc85588574)

[7) Palindrome. 44](#_Toc85588575)

[8) Prime Number 45](#_Toc85588576)

[9) Repeat Addition Quiz. 46](#_Toc85588577)

[11) Result. 49](#_Toc85588578)

[12) Test Break. 51](#_Toc85588579)

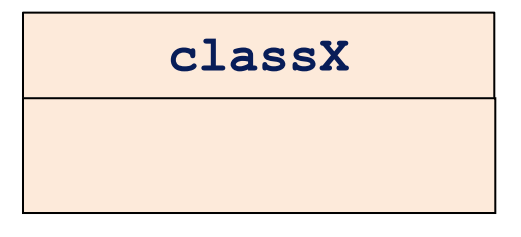
[13) Test Continue. 52](#_Toc85588580)

[14) Test Do While. 53](#_Toc85588581)

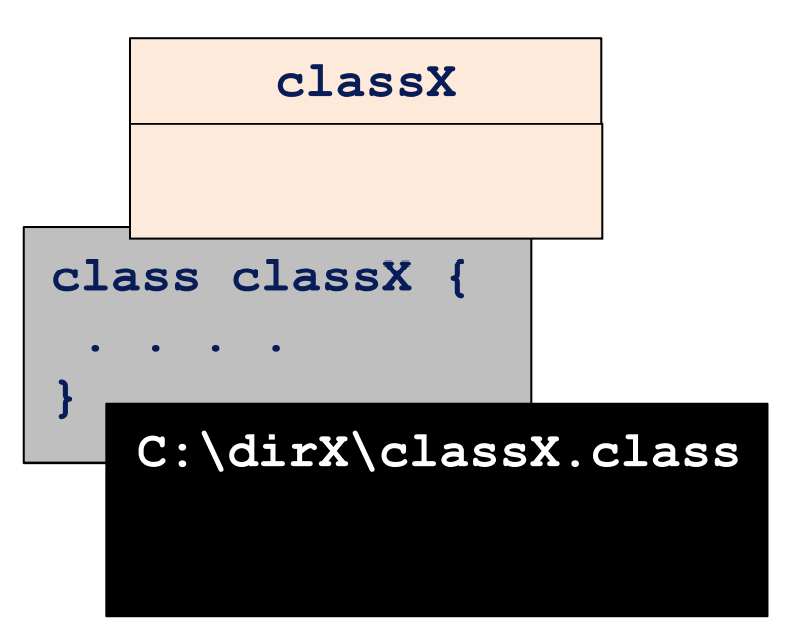
[15) Test Sum. 54](#_Toc85588582)

# 1) Java Class as a Basic Java Code Unit.

* Java class is the most elementary unit of standalone Java code (example: class javaX{…} ) that may exist independently in the OS-FS as a file, (e.g., classX.java in source code or classX.class as a Byte-code JVM-object code file).



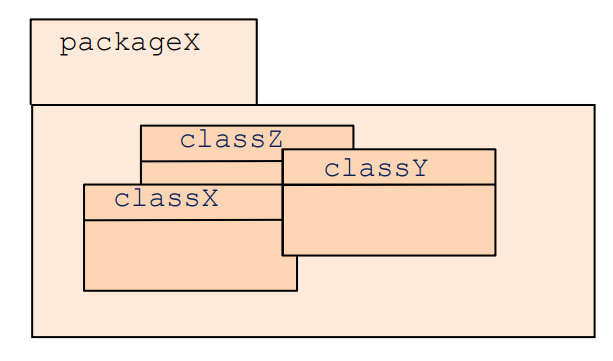
* Three elementary Java program views are:
* UML – design view.
* Java source code – file internal view, and
* OS-FS complied file, SA or installation view.



# 2) Packages.

* Packages are OOD and OOP entities used to group functionally related classes under one abstraction label/name.
* Package name (e.g., java) appear as dot separated from the content element, (e.g., lang)

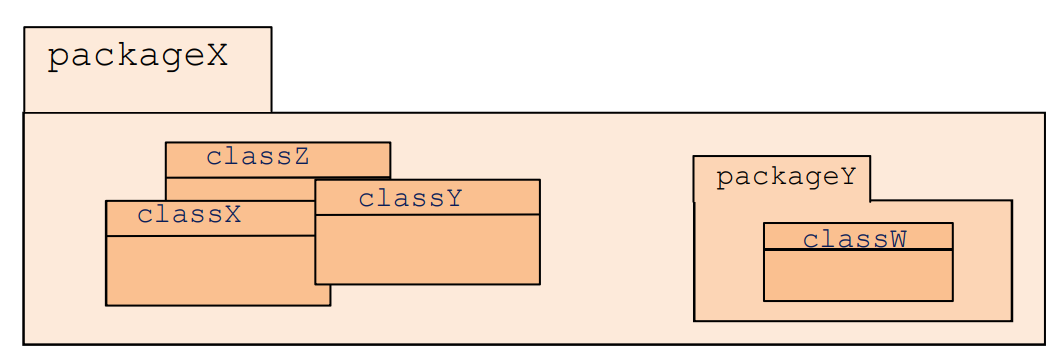
java.lang



# 3) Package Syntax – Name Scope.

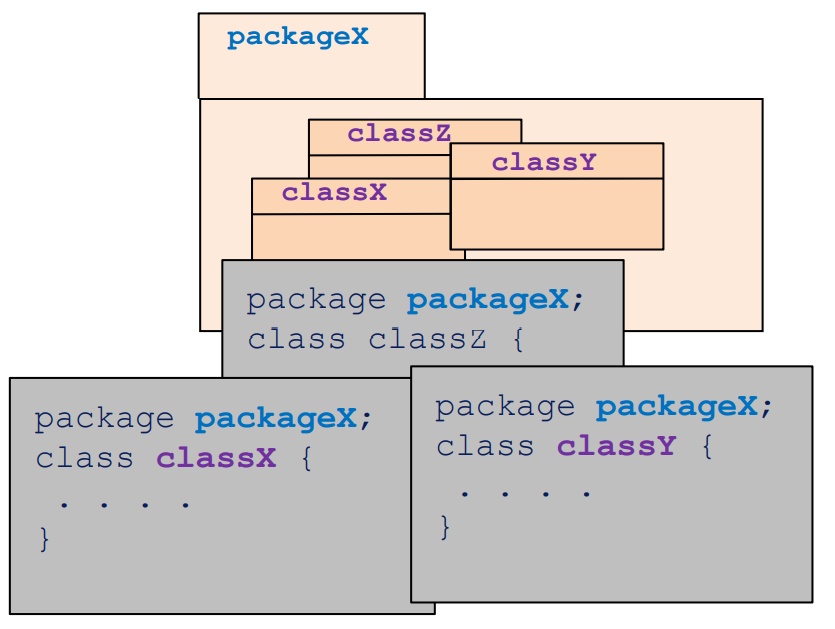
## Java provides a syntax for package names.

* Package names are separated by periods (e.g., packagesX.packageY.classW).
* Packages can contain classes or sub-packages, (e.g, package.classX or package.packageY).



# 4) Java Package as Code Organizational Unit.

* Java classes are organized in sets of relevant classes (semantically relevant, common-purpose, meaning or use relevant) called packages.
* A package is names container made of:
* Classes,
* Interfaces and
* Sub – packages.



# 5) Packages and Name Space Collision.

* As a container, package resembles and corresponds to the directory in the external OSFS directory tree structure.
* Packaging is a code organization tool to avoid name space collision. – There cannot be two classes with same name in a same Package but two packages can have a class with same name.
* Fully qualified class names precisely define class belonging.

java.lang.String

java.util.Arrays

java.io.BufferedReader

java.util.Date

# 6) Question: Packages.

## What is the meaning of name space collision?

Answer)

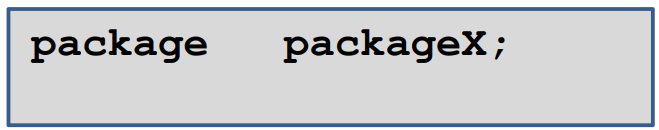
* Name space collision is an event when 2 different classes with the same name appear and JVM or compiler may not be able to decide which one to use.

Aclass in collision with another Aclass

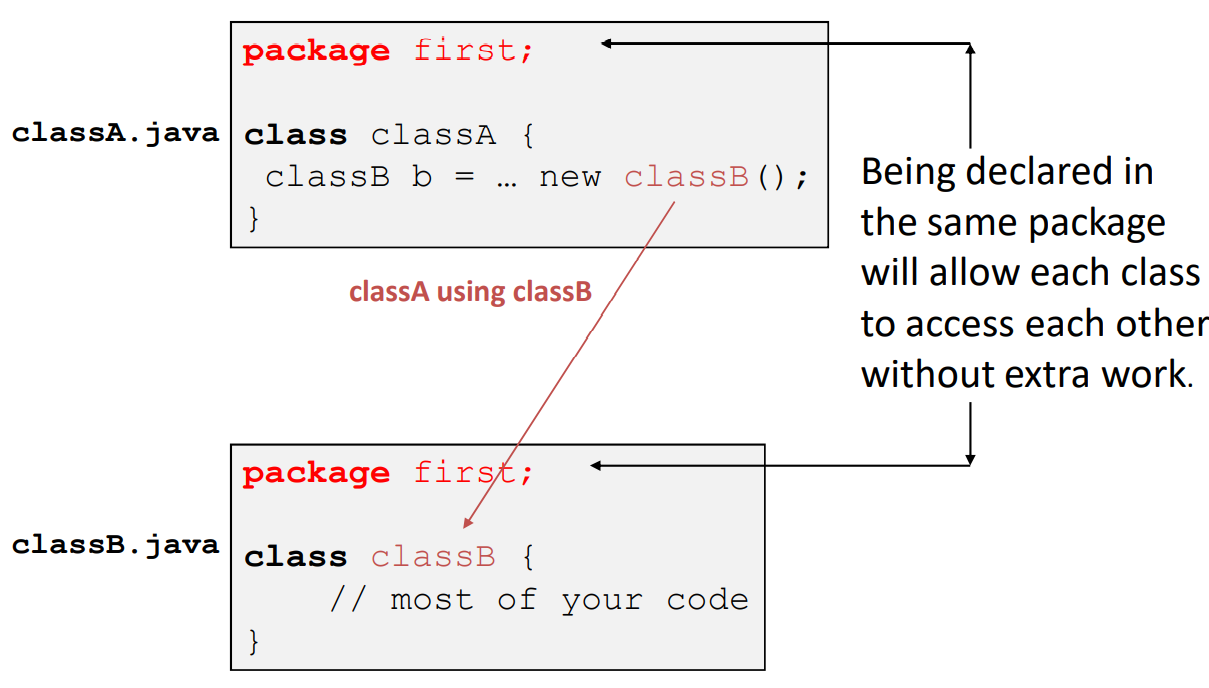
java.packA.Aclass is not in collision with another java.packB.Aclass

# 7) Adding to the Package.

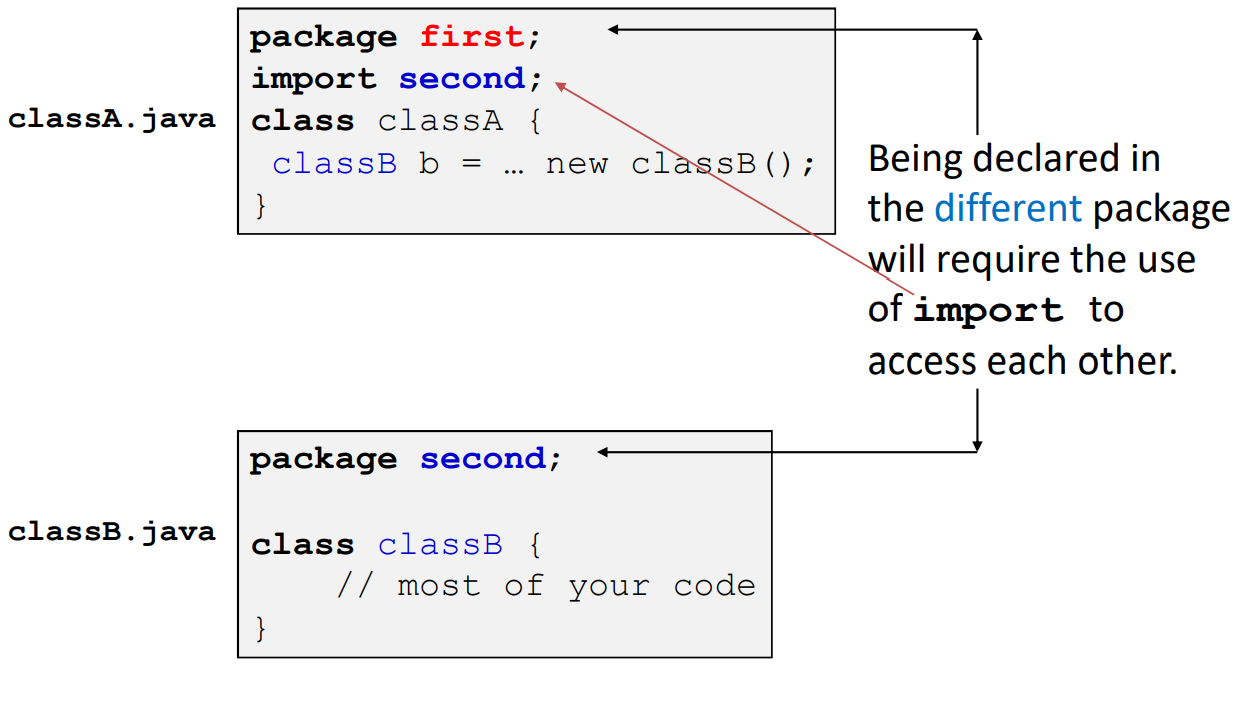
## At OOP/compile-time, to have class under development belong to a particular package use Java package operator.



# 8) Multiple Classes in the Package first.



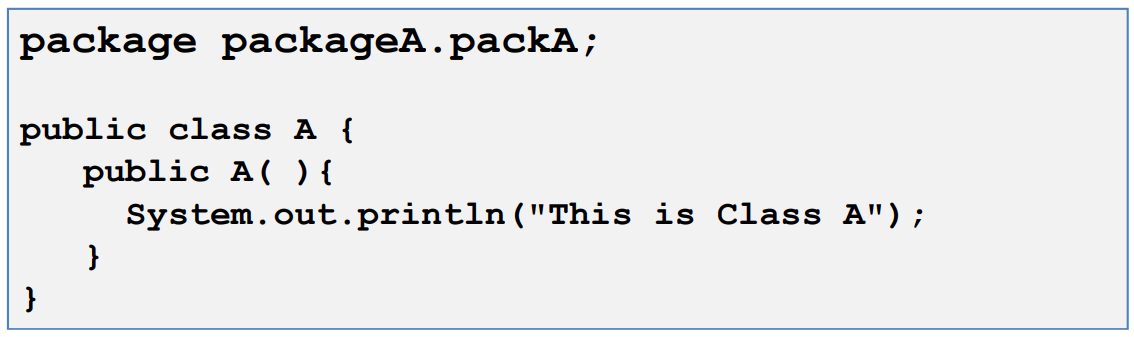
# 9) Two classes in Different Packages.



# 10) Making a Class Belong to a Package.

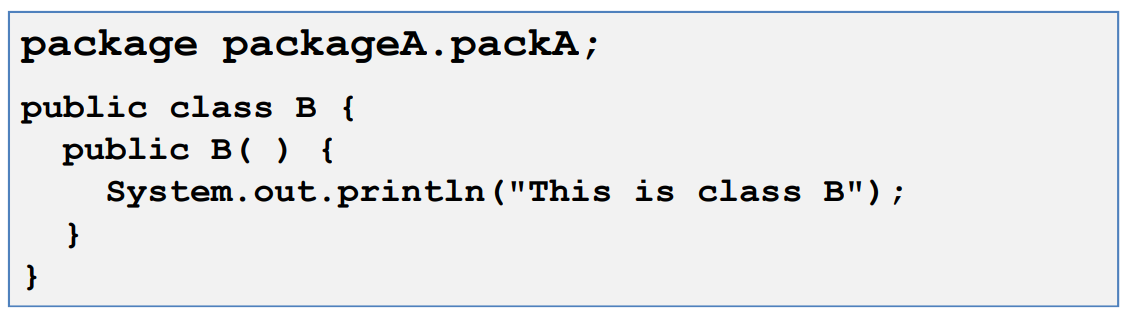
## Include a proper package statement as first line in source file.

* Make a class A belong to packA.



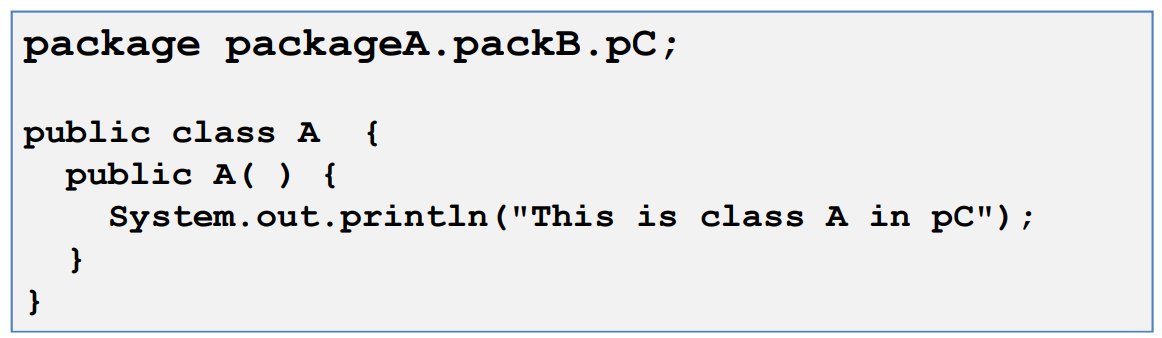
* Name the source file as A.java compile it and store the A.class file in packA directory.

## Make class B belong to packA



* Name the source file as B.java and compile it and store the B.class file in packA directory.

## Make class A belong to package pC:



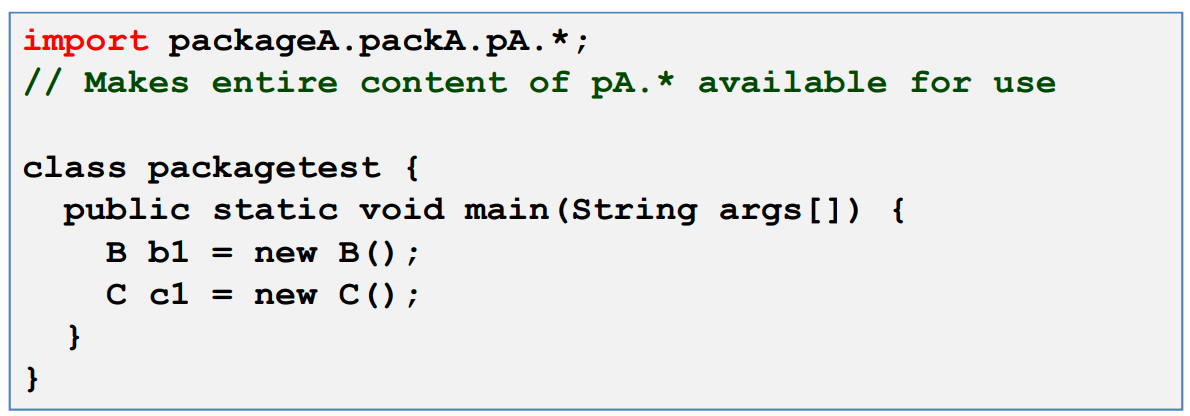
* Name the source file as A.java and compile it and store the A.class file in pC directory.

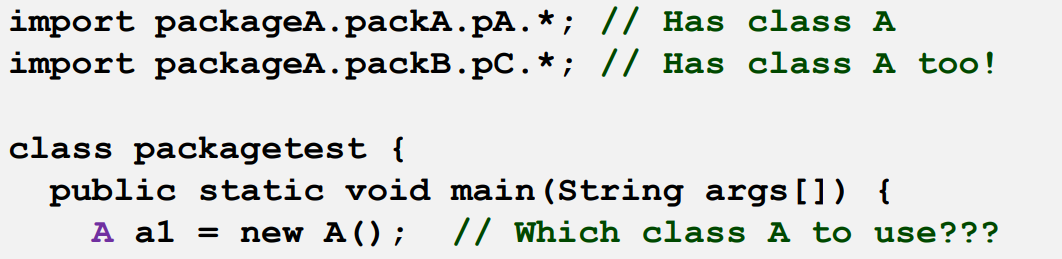
# 11) Importing the package.

## Import statement allows importing or linking of a package.

* Java Library packages are automatically imported irrespective of the location of compiling and executing program.

# 12) Importing or Linking packX.packA.pA;





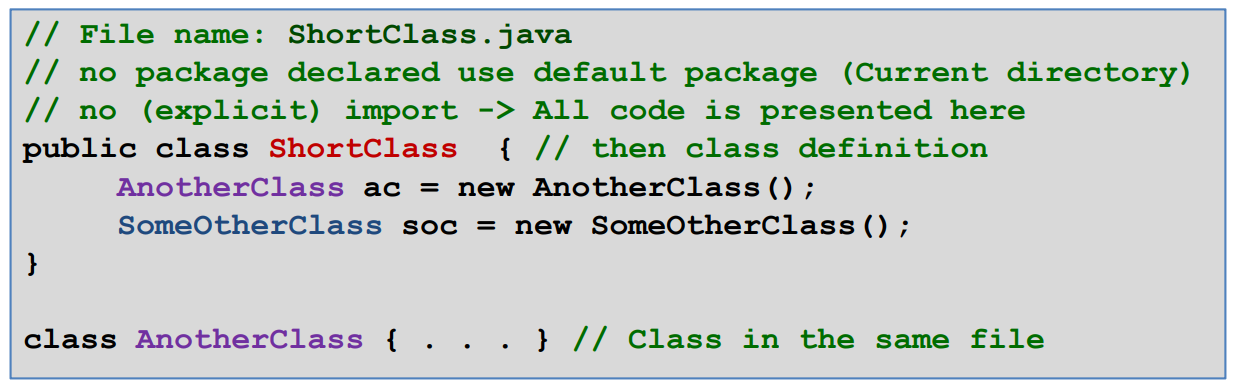
* Class A is present in both imported packages pA and pC.
* Class A has to be fully qualified in this case.

packageA.packA.pA.A a1 = new packageA.packA.pA.A();

or

packageA.packB.pC.A a1 = new packageA.packB.pC.A();

# 13) Default Package.



* ShortClass class resides in file ShortClass.java
* ShortClass does not explicitly declare itself as the member of any package so it is implicitly placed in the default package of classes in the current directory.
* ShortClass does not need to import classes from other packages because it is referencing two other classes since:

– AnotherClass is defined in the same file as ShortClass

– SomeOtherClass is defined in the file located in the same, local directory as ShortClass.java file so we do not need to import it.

# 14) Default Package Convenience.

## The use of default packages (current directory) is convenient.

– Usually when you start developing classes you do not like to worry about locations of other used classes, as long as they are together in the same current directory or in default directories defined by the path environment variables.

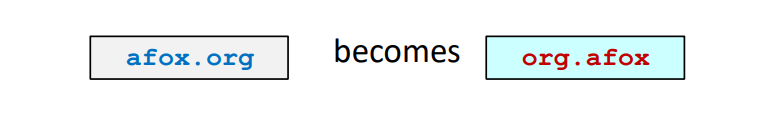
– After you have your class code well debugged, you may want to separate classes into packages according to some relationship they have with each other and some plans for future reuse. Code modification is simple; only first file lines have to be modified

# 15) Java and a Global Namespace.

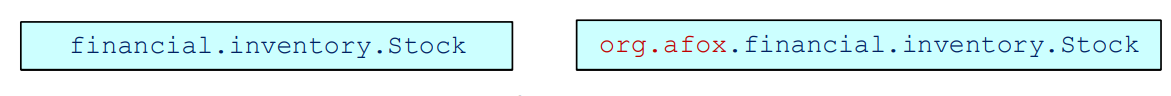
## • The designers of Java have proposed an Internet-wide, i.e., Web-wide unique package naming scheme.

─ It includes the Internet DNS name of the organization within which the package was developed.

─ The domain is reversed and the sub-domains become packages.



- The domain portion is prepended to any package developed within afox.org.



becomes

- This standard guarantees that no other organization will create a class whose name conflicts with the classes developed with afox.org.

# 16) Putting Classes in Packages.

## • To put a class into a package, one uses the "package" statement.

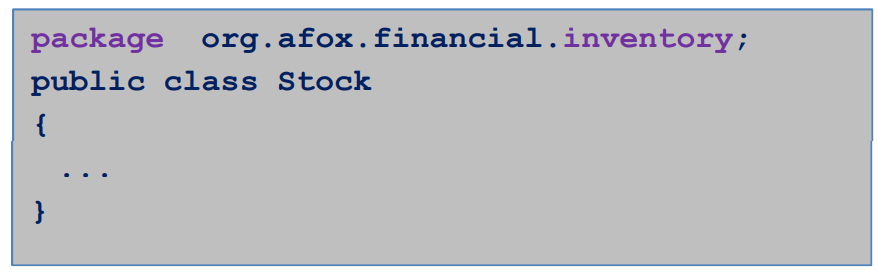
- The package statement MUST be the first line of code within the file.

- It can be proceeded by comments.

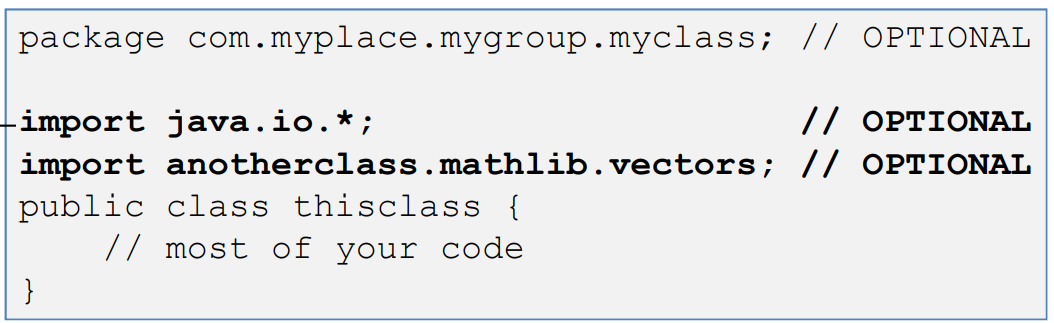
• If no package statement is supplied, the class is placed in the “default” or current package.

- The default package is locally a package with no name.

Example: In file Stock.java:



# 17) Java and a Global Namespace.

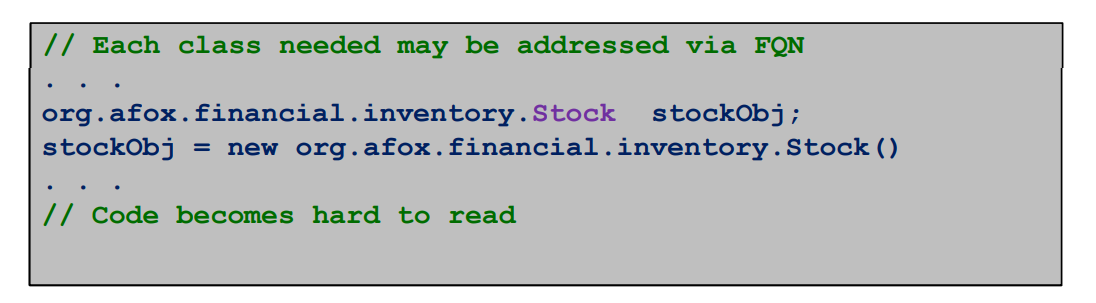


* Directly related to the packages.
* Used to provide a shorthand.
* Used for accessing OTHER classes NOT related to the package statement (if it exists in a file).
* CAN have import without package and vice versa.
* While they are related, the use of the use of them in the SAME file is typically independent/coincidental.
* Package defines the package THIS class in IN.
* Import used to ease USE of the classes in OTHER packages.

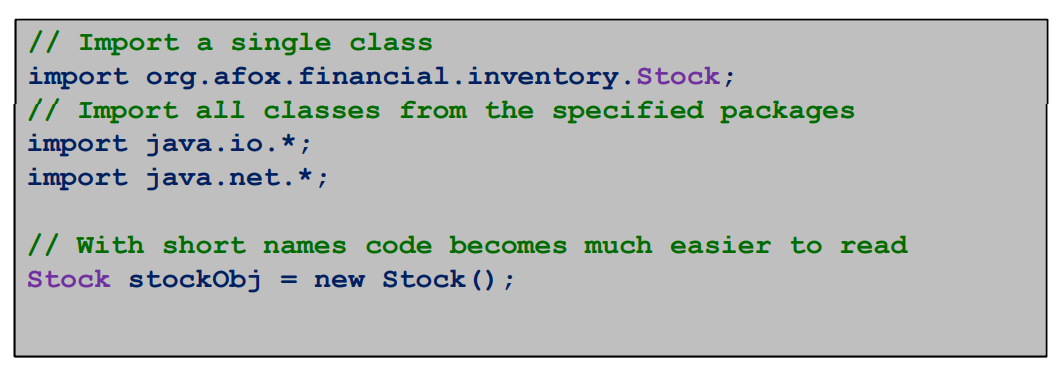
# 18) Using import to Simplify Code.

## Unfortunately, the proposed naming scheme creates very long names for classes.

* A class which was originally name “Stock” is now named “org.afox.financial.inventory.Stock”.
* This makes coding difficult. It dramatically reduces code readability.

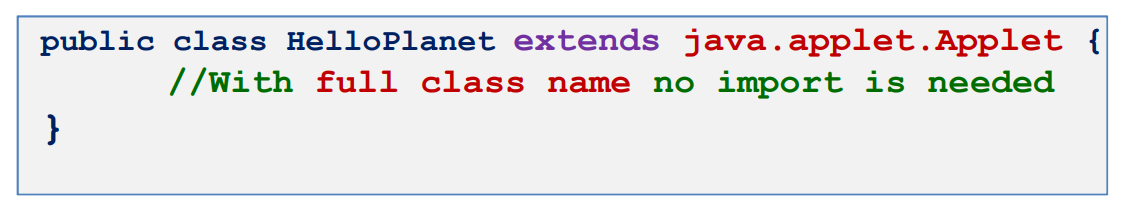


* To avoid having to use the fully qualified class name for all classes, the programmer can "import" some or all of the classes from a given package.
* Once a class has been imported, it may only be referenced by its short name "Stock" instead of “org.afox.financial.inventory.Stock”.

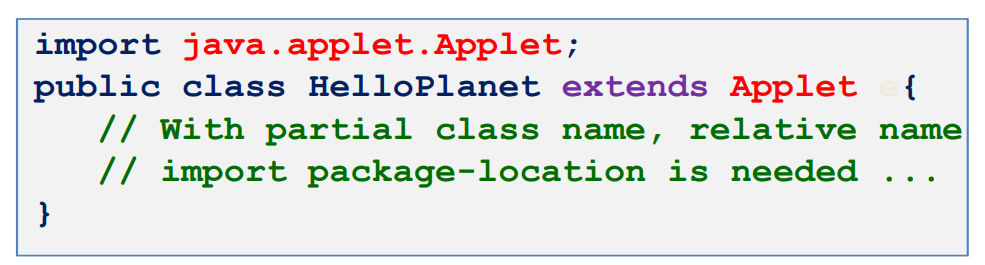


# 19) Package Inclusion Simplifies Code Text.

* The import statement can make an individual class name available in a compilation unit. For example, instead of:



* We may write



# 20) Import statement.

## • Options for the import statement:

- blank

- no import command and no packages are imported.

get java.lang.\*; by default

- import packageX;

- imports packageX

OSFS has to be able to locate the directory in which this package resides.

# 21) Importing the Package at Run Time.

## • At run time JRE looks at two places for user created packages:

1. Under the current working directory
2. At the location specified by CLASSPATH environment variable.

• Most ideal location for compiling/executing/testing a program is immediately above the package directory structure.

# 22) CLASSPATH Environmental Variables.

* CLASSPATH Environmental Variable defines OSFS search path for the location of the root directory of the package hierarchy.

# 23) Question: CLASSPATH Environment Variables.

* Consider the following statement:

package mypack;

## What should be true in order for your program to find package mypack?

## Answer:

* Program should be executed from the location immediately above mypack directory (Directory mypack is a subdirectory of the current directory),

or

* mypack directory full path name should be listed in the set of directories for CLASSPATH.

# 24) Resolving Packages at Runtime.

## • Java compiler compiles .java files to .class files.

• Each compiled class resides within its own .class file.

• Classes are loaded into the virtual machine by the ClassLoader

• Because of packages, .class files can now exist within a large number of directories

• For performance reasons, the ClassLoader must have a quick way of resolving a fully qualified class reference to .class file.

• Resolving packages to directories is simple.

• However, the virtual machine must know where to find the directory structure.

• The ClassLoader uses an environment variable called CLASSPATH to find the directory which contains the package directories.

# 25) Is CLASSPATH important at Compile Time?

## • CLASSPATH is also used by the compiler to resolve association issues at compile time:

• If an object sends a message to another object, the compiler must ensure that the receiving object will be able to respond to the message.

• The receiving object's class must be loaded by the compiler and checked.

• If the receiving objects class has already been compiled, the above process works.

## • However, what if the class has not yet been compiled and no .class file exists?

• The compiler attempts to locate the receiving objects .java file (using CLASSPATH) in an attempt to compile it.

• This means your source file structure must also match the package directory structure.

• The root of your source tree must also be within your CLASSPATH.

# 26) Compiling Classes with Packages.

## Packages provide a logical namespace for classes

* When compiled, the package structure is physically represented using a directory structure
* The .class file for the class “org.afox.financial.inventory.Stock” will reside at the following path under the CLASSPATH:

org/afox/financial/inventory/Stock.class

* Package names map directly to subdirectories.
* The programmer can externally specify/custom-change the root of the package directory structure using the -d option with the javac compiler.



* Will cause the compiler to place the .class file at:

/home/schock/classes/org/afox/financial/inventory/Stock.class

# 27) .jar files as Java Packages.

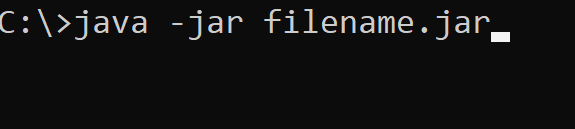
Extraction of certain filex.class from the archives packagesX.jar requires execution of a command:

jar -xf fileX.class packageX.jar

# 28) .jar File Containing the Main Class.

## JVM class loader checks specified packages by a programmer and using jar –x extracts needed classes.

* Jar files can be defined to contain a main class.
* JVM can be given a jar file instead of a main class file (program driver class) to stat certain program execution.
* If a .jar file has a main class defined, it can be executed using the following command:



Work from [Microsoft PowerPoint - 725\_03\_Lecture\_03\_ClassesPayh.pptx (tfbor.com)](http://tfbor.com/02_725/03_ObjctClassPackge/725_03_Lecture_02_CLASSPATH.pdf)

Java Software Engineering.

# 29) Defining Abstraction.

## Abstraction is the process of recognizing and extraction common features from a set of specific example-objects/samples.

* Abstraction is a process of:
* Defining/distinction of general/essential/conceptual/general-attributes and properties while.
* Ignoring the specific/variable/inessential detail-attributes and properties.

# 30) Different types of abstraction.

## Data Abstraction

Programming languages define constructs to simplify the way data is presented to the programmer.

## Functional Abstraction

Programming languages have constructs that ‘gift-wrap’ very complex and low level instructions into instructions that are much more reusable and readable.

## Object Abstraction

OOP languages take the concept even further and abstract programming constructs as objects.

# 31) Defining a Java Class.

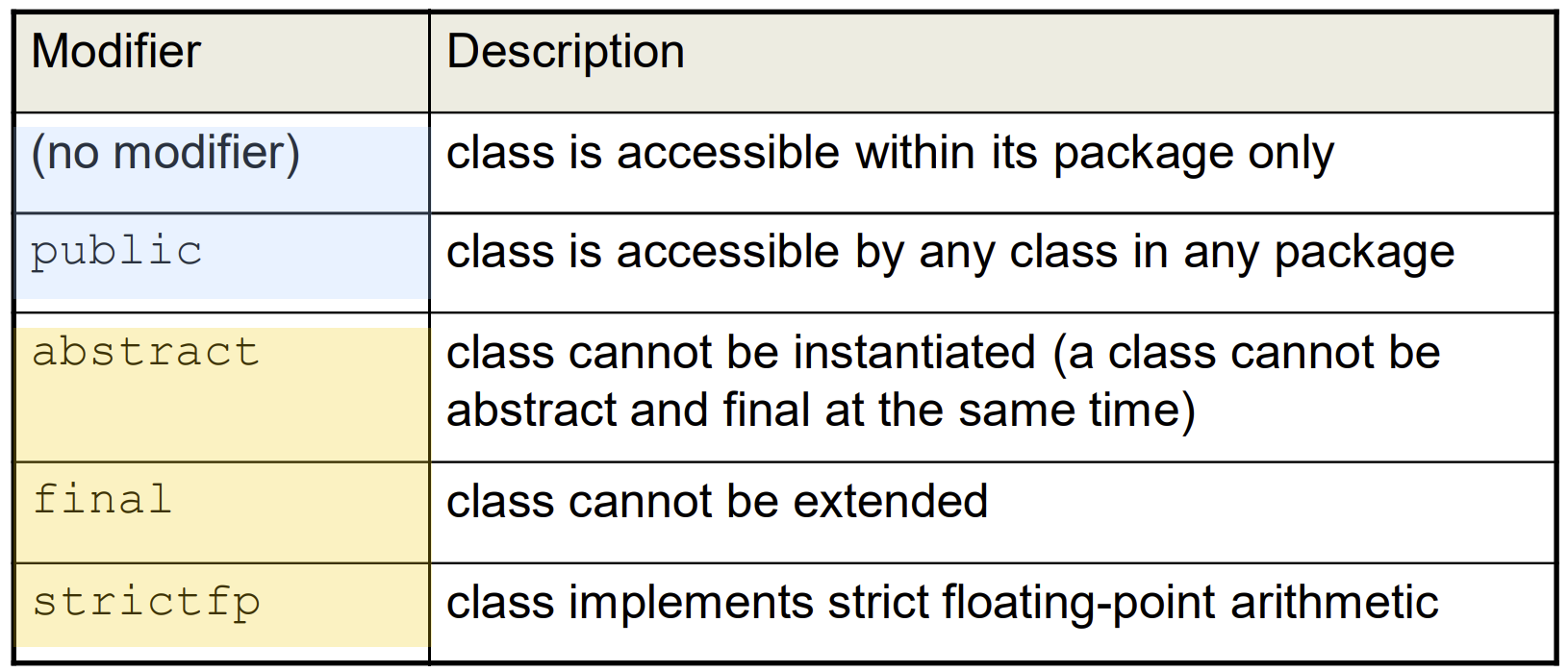
## A Java Class denotes a category of objects and acts as a blueprint for creating such objects.

* It defines its members referred to as fields and methods.
* The fields (also known as operations) define activities or behaviors that the class exhibits.

# Class Modifiers.

## Class modifiers change the way a class can be used.

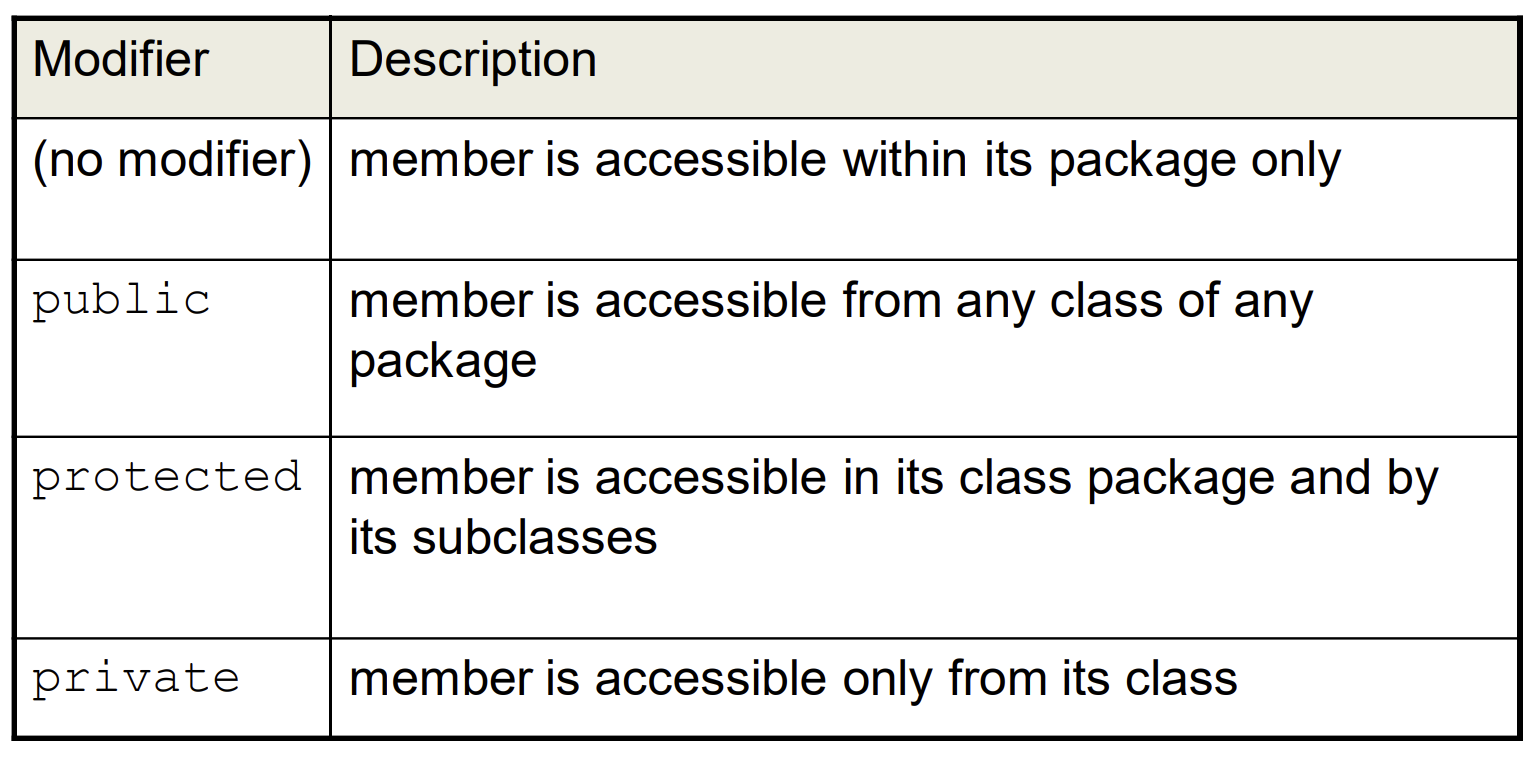
* Access modifiers describe how a class can be accessed.
* Non-access modifiers describe how a class can be manipulated.

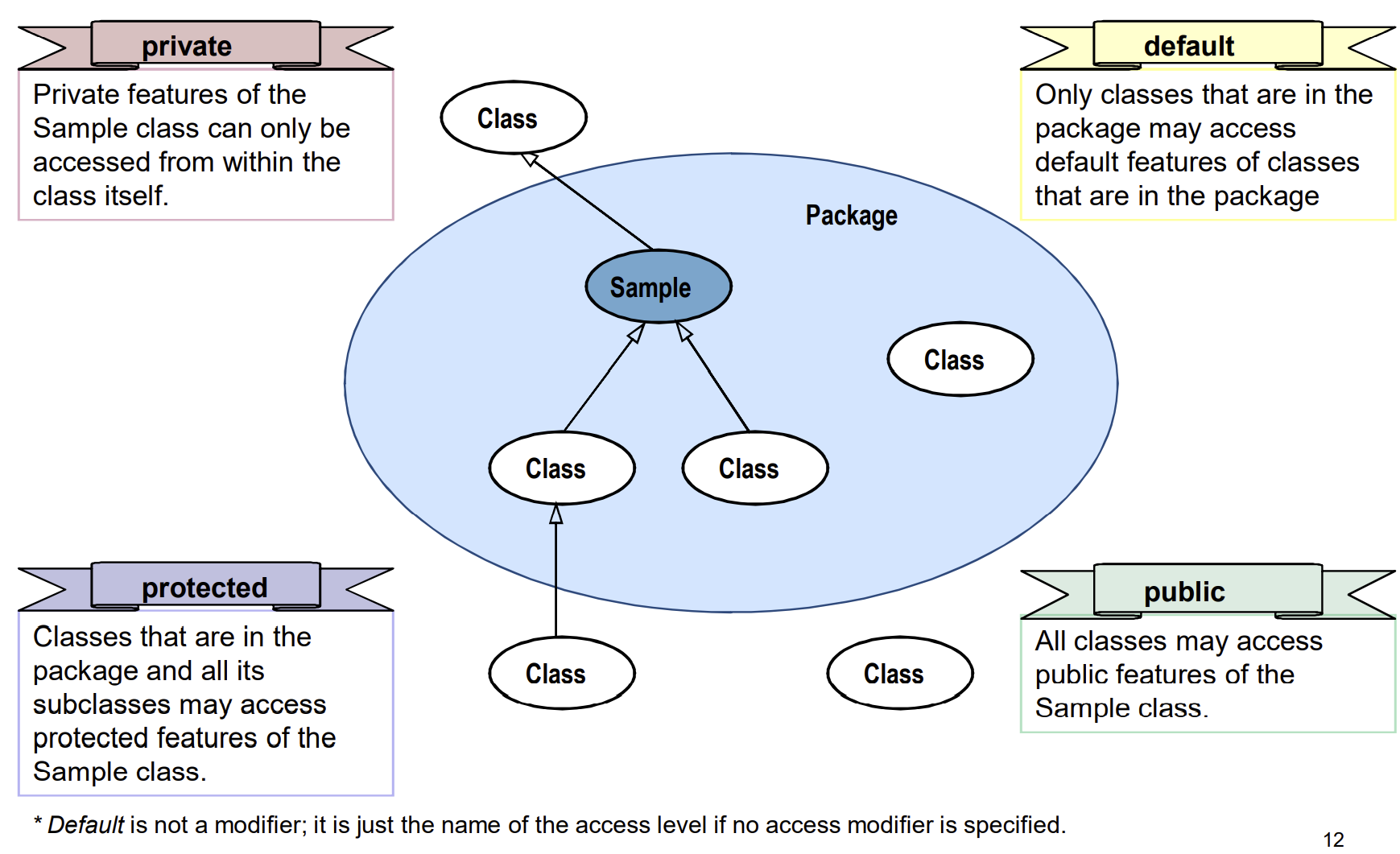


# Access Modifiers.

## Member modifiers change the way class members can be used.

* Access modifiers describe how a member can be accessed.

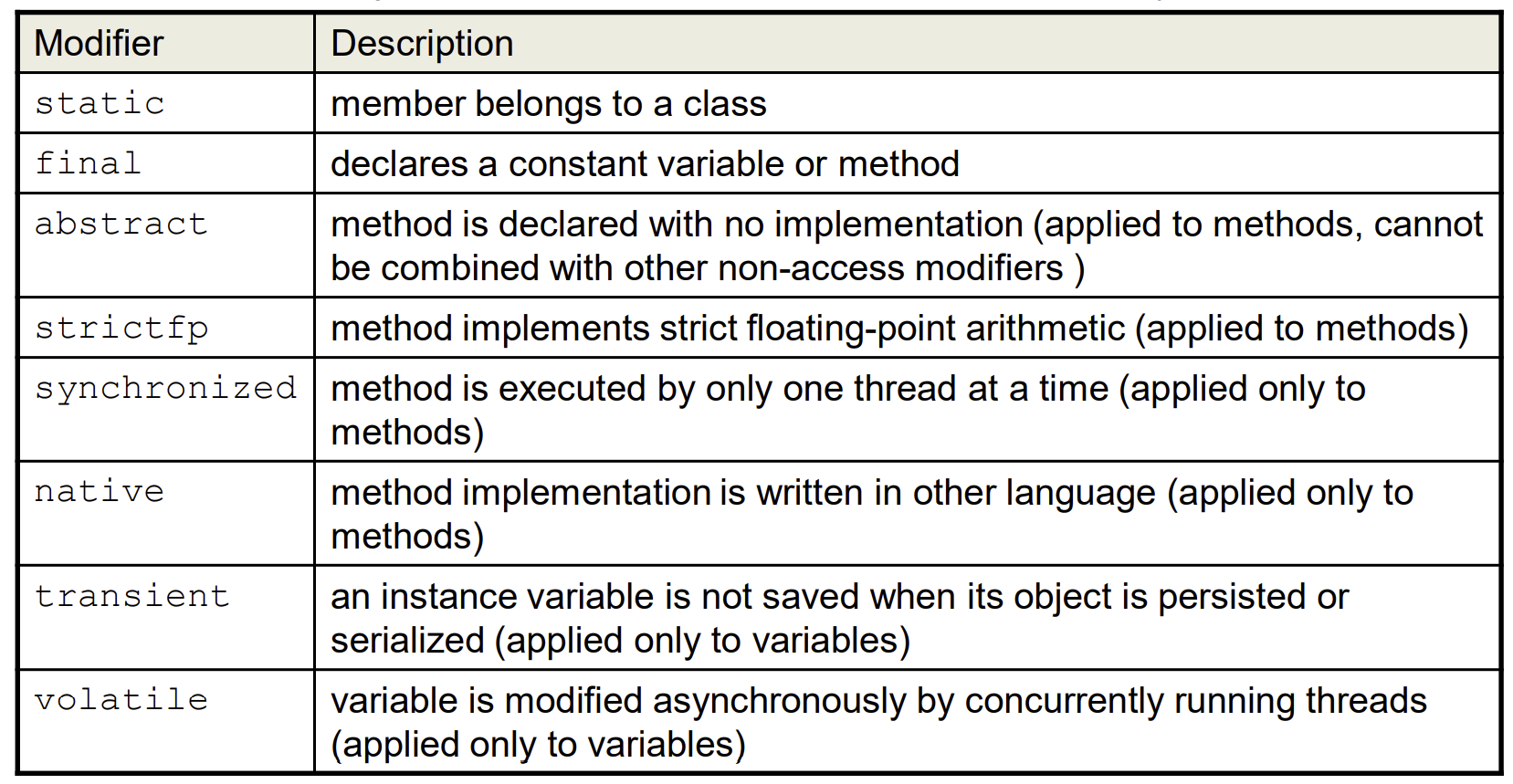




# Member Modifiers.

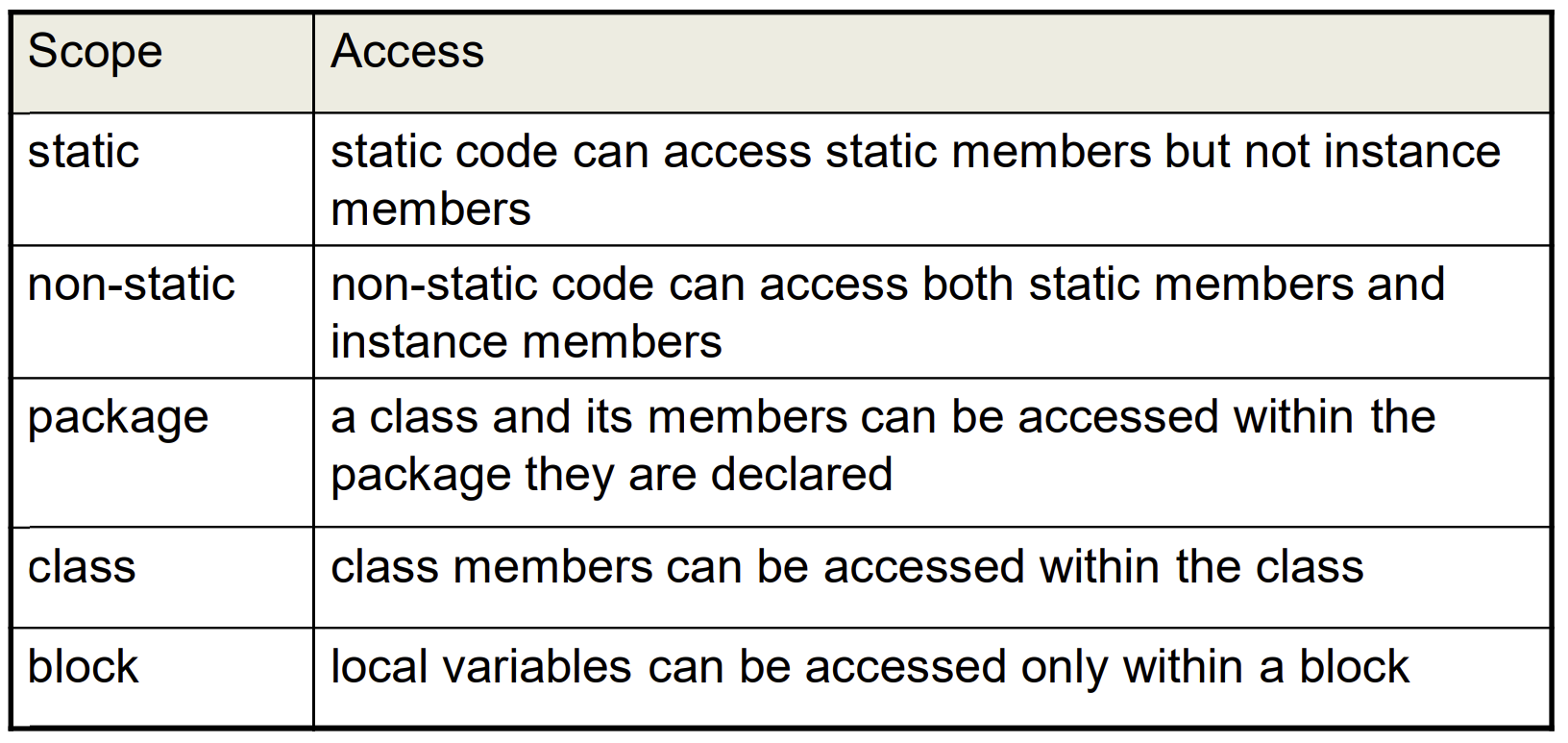
## Member modifiers change the way class members can be used.

* Non-access modifiers describe how a member can be manipulated.



# Accessibility Scope.

## Accessibility scope defines the boundary of access to a class and its members



# Defining Encapsulation.

## Encapsulation is the process of hiding an object’s implementation from another object, while presenting only the interfaces that should be visible.

# Encapsulating a Class.

## Members of a class must always be declared with the minimum level of visibility.

* Provide setters and getters (also known as accessors/mutators) to allow controlled access to private data.
* Provide other public methods (known as interfaces) that other objects must adhere to in order to interact with the object.

# Setters and Getters.

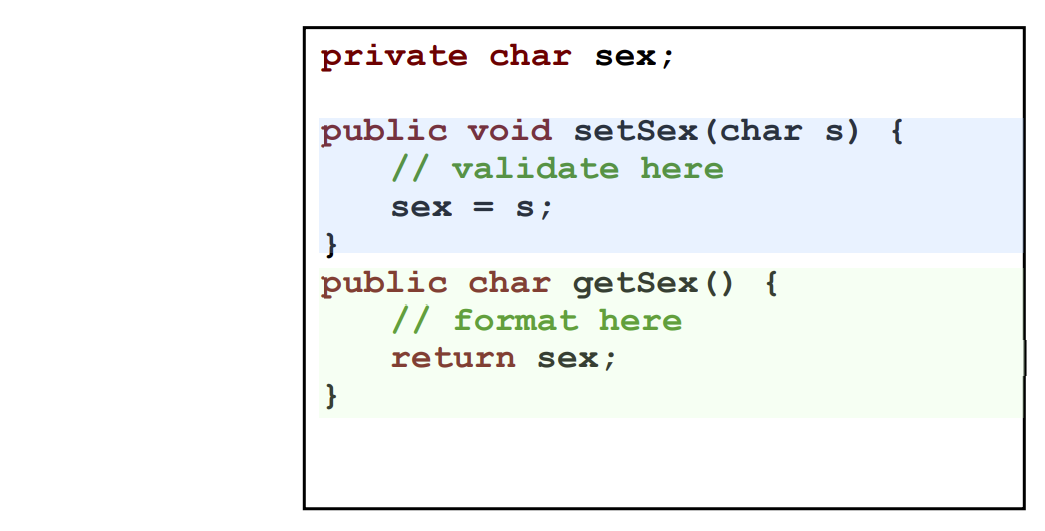
## Setters and Getters allow controlled access to class data.

* Setters are methods that (only) alter the state of an object

– Use setters to validate data before changing the object state.

* Getters are methods that (only) return information about the state of an object.

– Use getters to format data before returning the object’s state.



Work from [Microsoft PowerPoint - \_735\_03\_Lecture\_1\_Tar\_Jar\_v1.pptx (tfbor.com)](http://tfbor.com/02_725/03_ObjctClassPackge/725_03_Lecture_03_JAR.pdf)

Basic Archives Procedures.

# Basic Archives Procedures Overview.

• In order to back up or easily transfer files via ftp or another method, SA needs to be able to package and restore the files easily.

• This module covers the following commands:

− jar

− tar

− compress

– mt

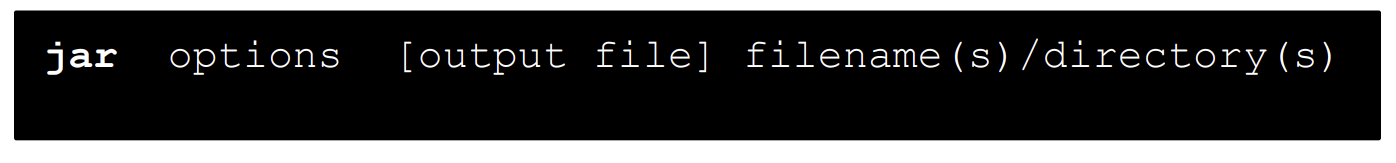
• In addition, specific instructions for backing up and restoring your home directory using tar and compress are included.

# The jar Command.

## The jar command is similar to the tar command, but compresses the resulting file in the same step.

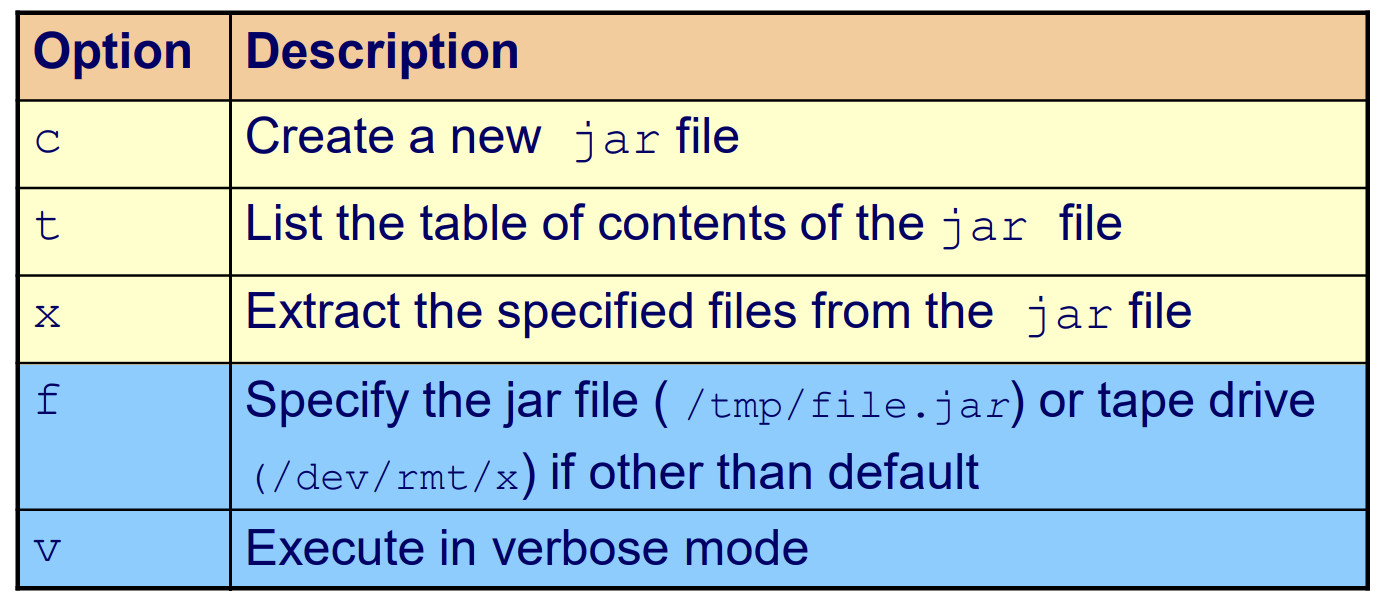
* It is a Java™ application that combines multiple files into a single JAR (Java archive) file.
* It is also a general-purpose archiving and compression tool, based on ZIP and the ZLIB compression format.
* The jar command was originally created for Java programmers to download multiple files with one request, rather than having to issue a download request for each separate file.
* jar is standard with the Solaris 7/8 operating system, but is available on any system that has Java virtual machine (JVM) installed.

## The syntax for the jar tool is almost identical to the syntax for the tar command.



* It is not necessary to use a hyphen (-) before options when issuing the jar command.

# The jar Command Option.



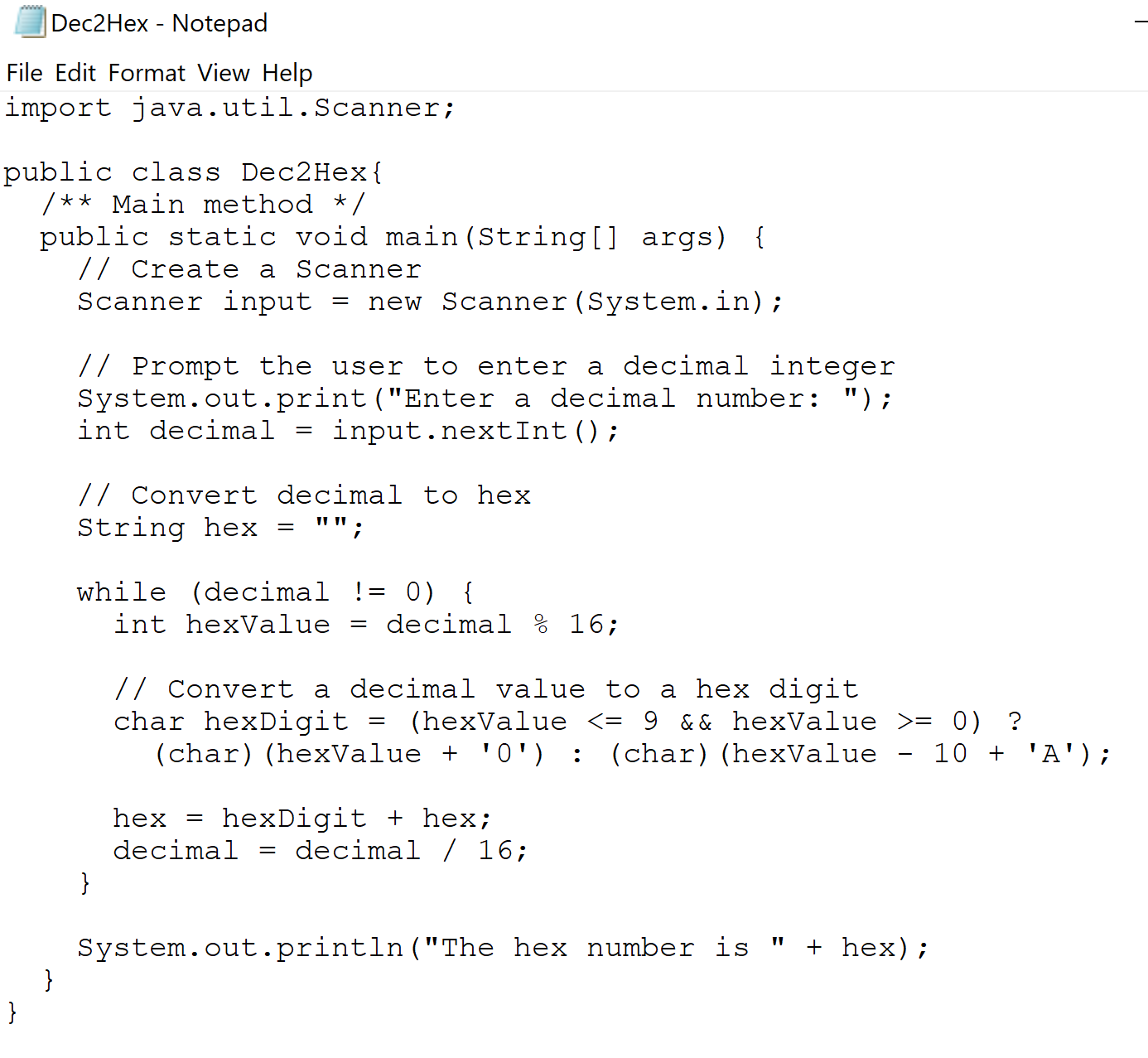
The jar tool is not intended for backing up symbolic links.

Error messages will display if this is attempted.

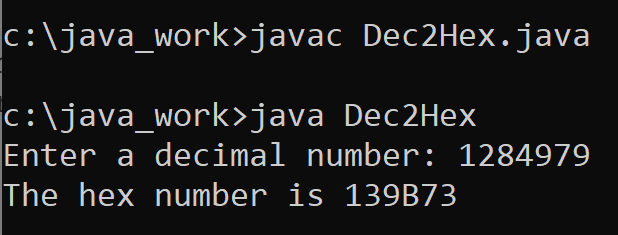
Work from [Index of /02\_725/03\_ObjctClassPackge/Homework\_book/chapter5 (tfbor.com)](http://tfbor.com/02_725/03_ObjctClassPackge/Homework_book/chapter5/)

# Chapter 5 Homework.

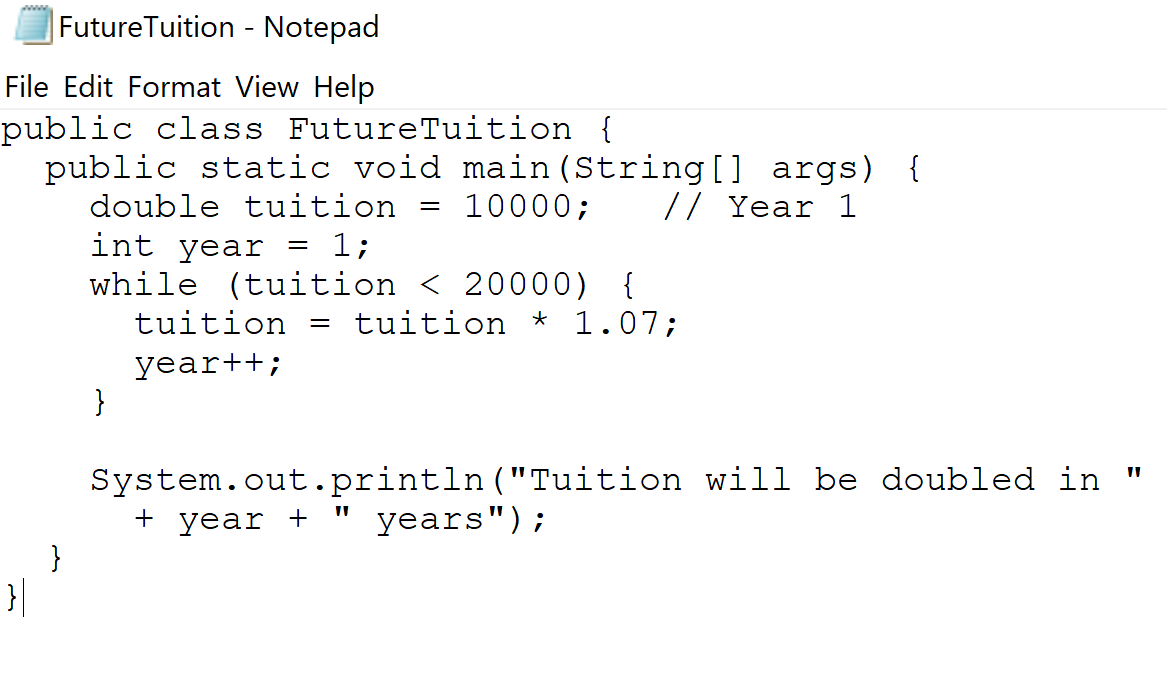
# 1) Decimal to Hex.



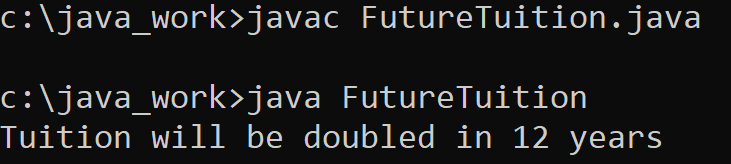
Result:



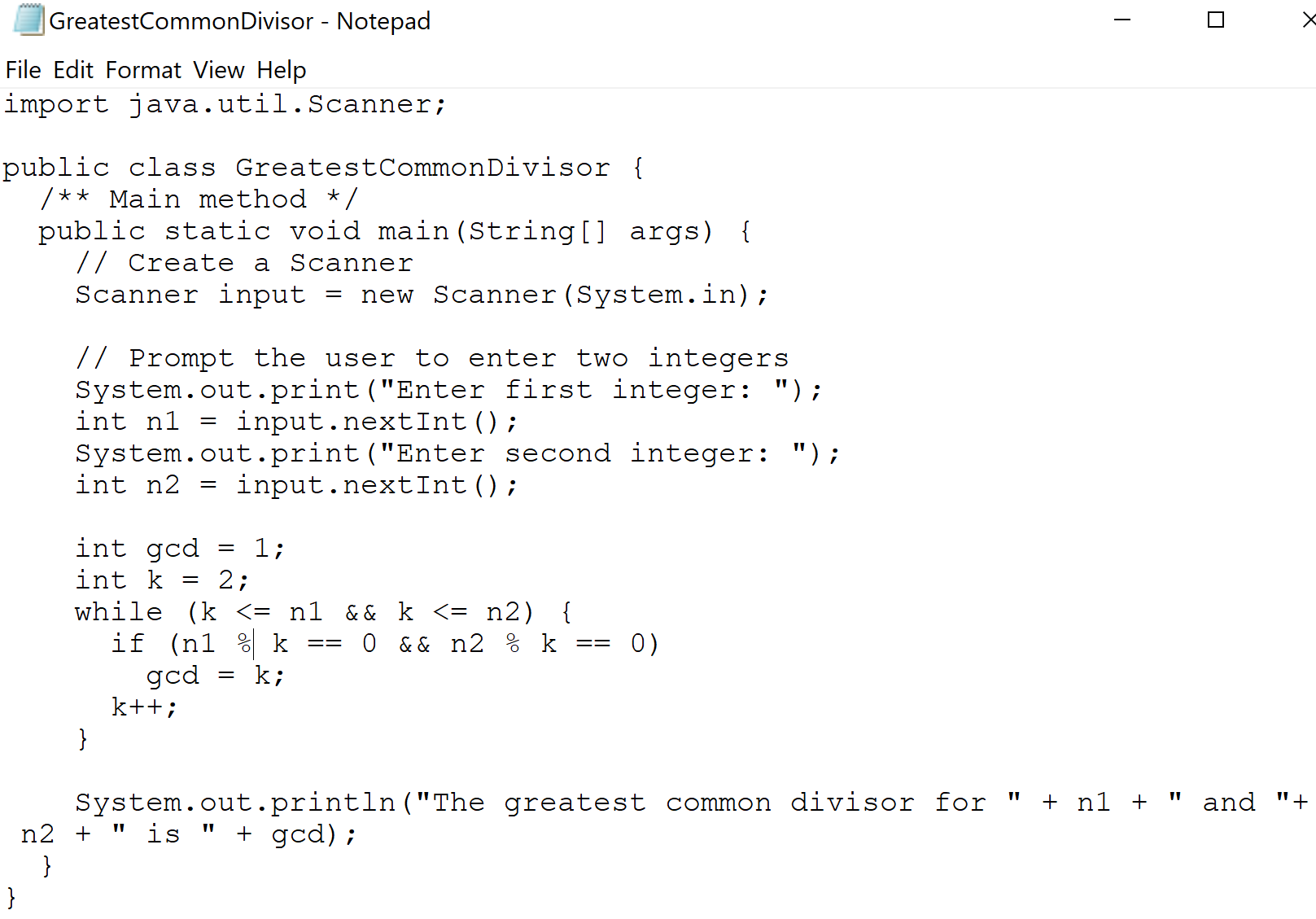
# 2) Future Tuition.



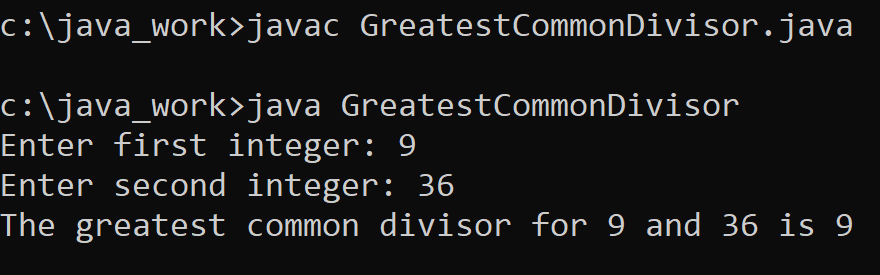
Result:



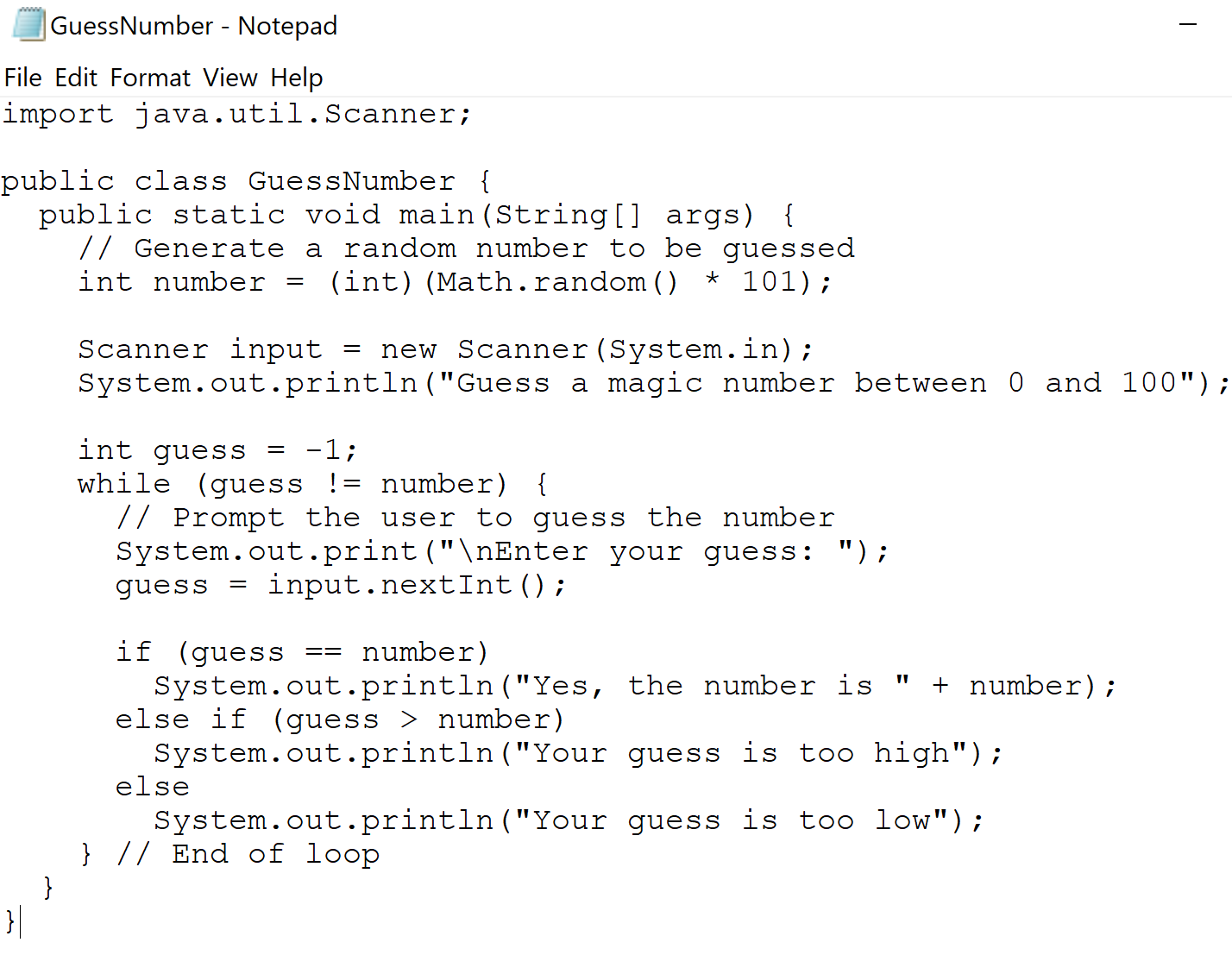
# 3) Greatest Common Divisor.



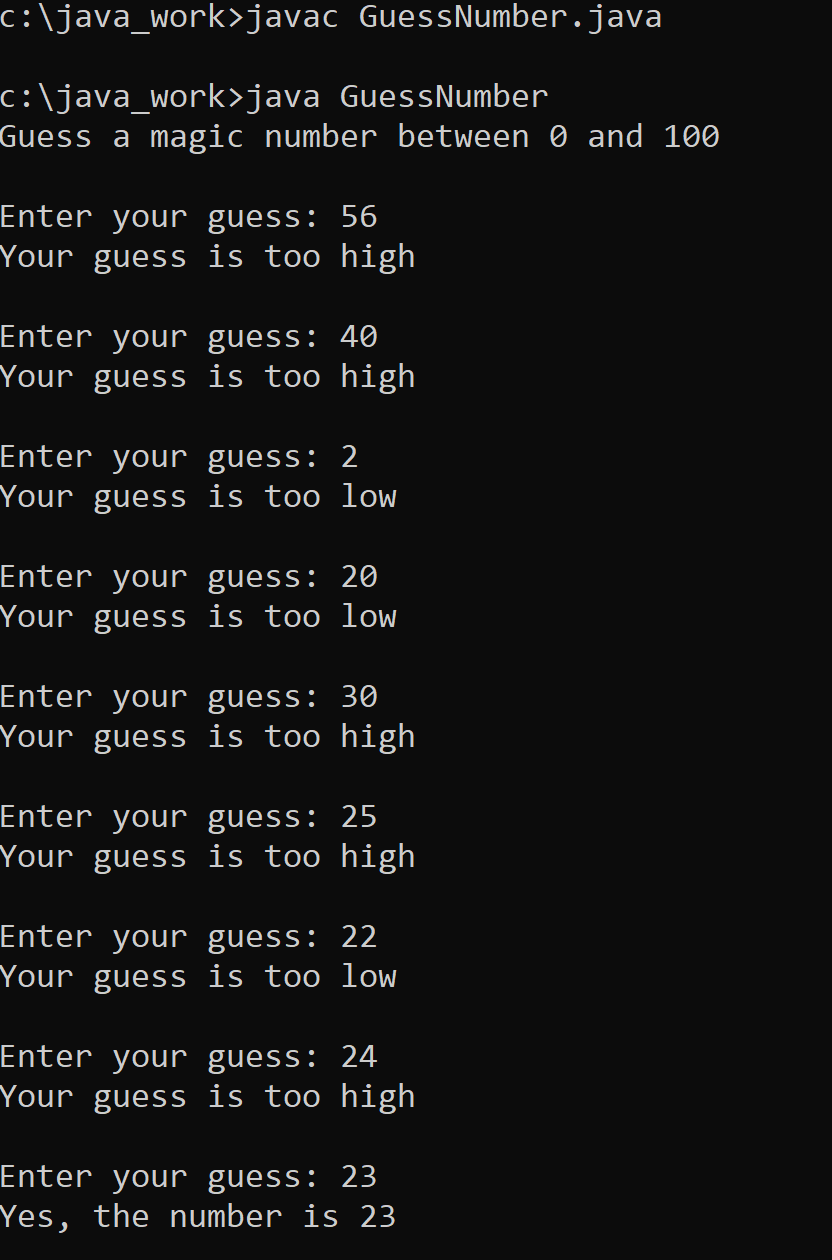
Result:



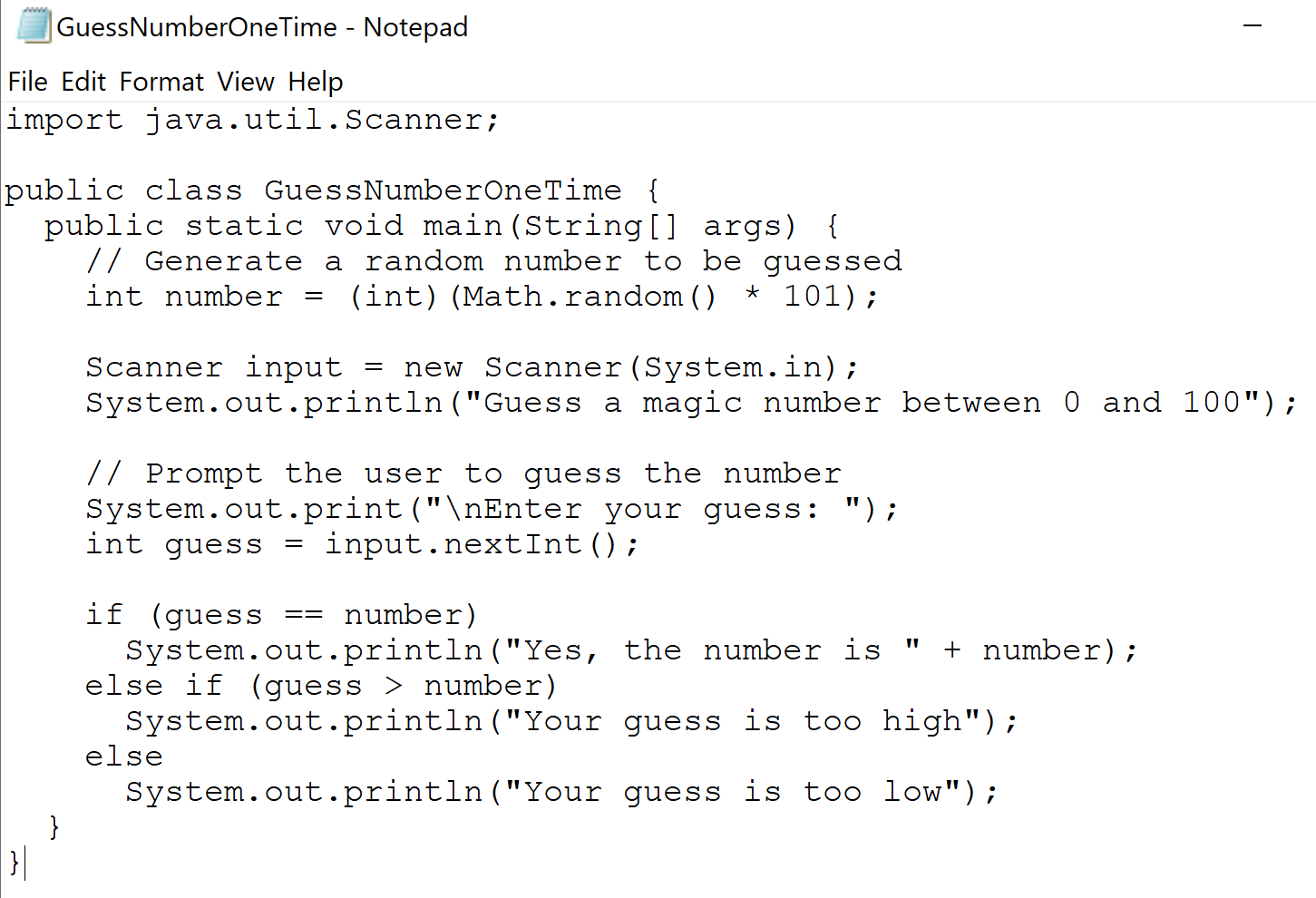
# 4) Guess Number.



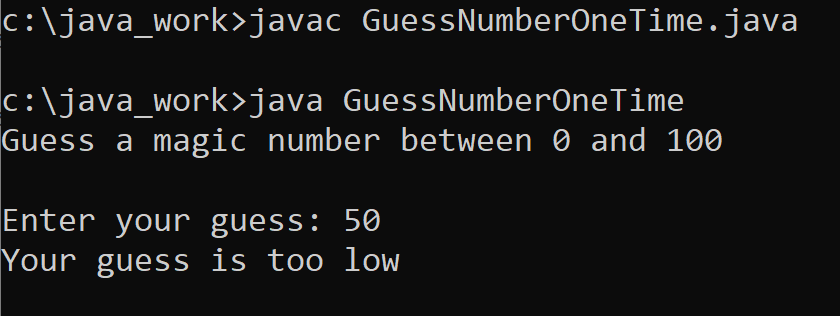
Result:



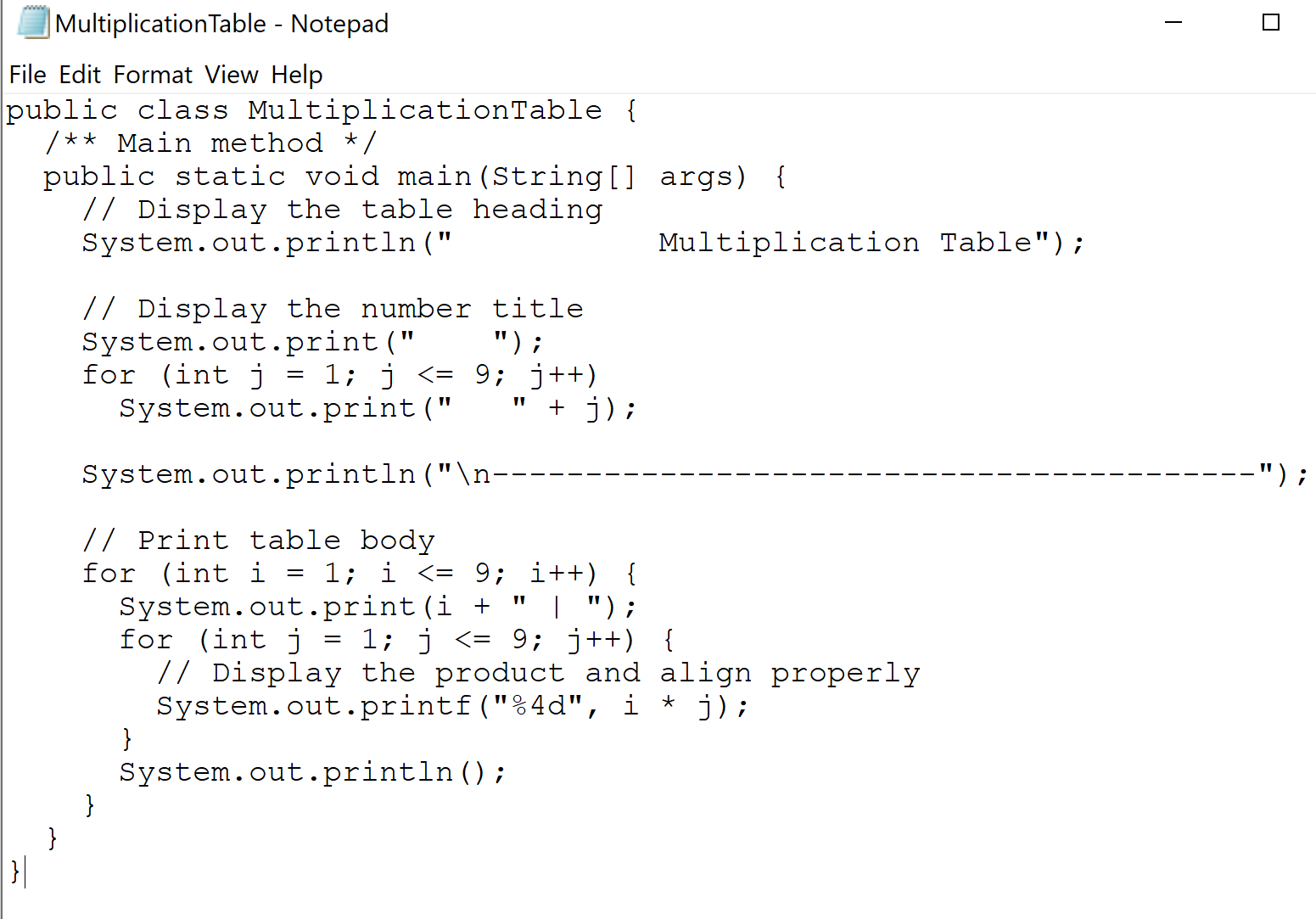
# 5) Guess Number One Time.



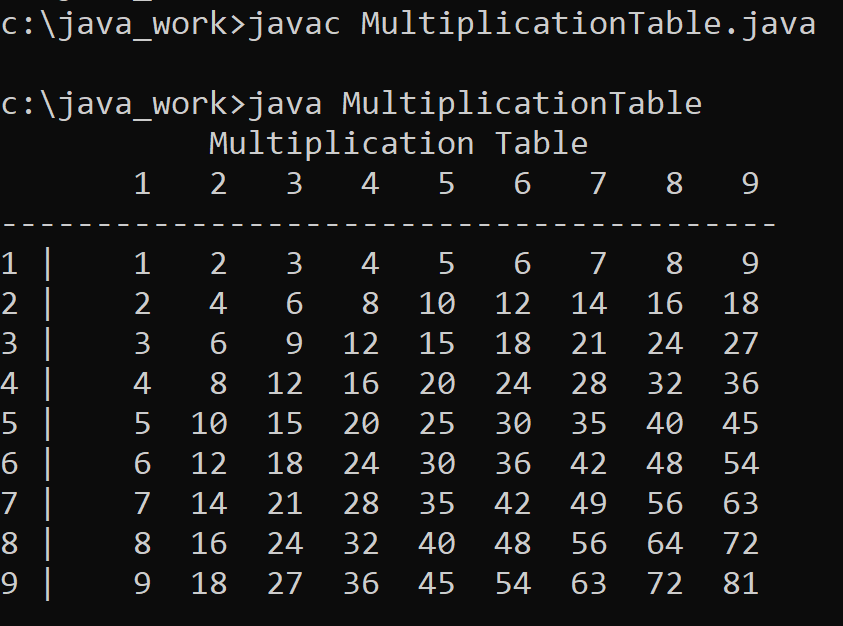
Result:



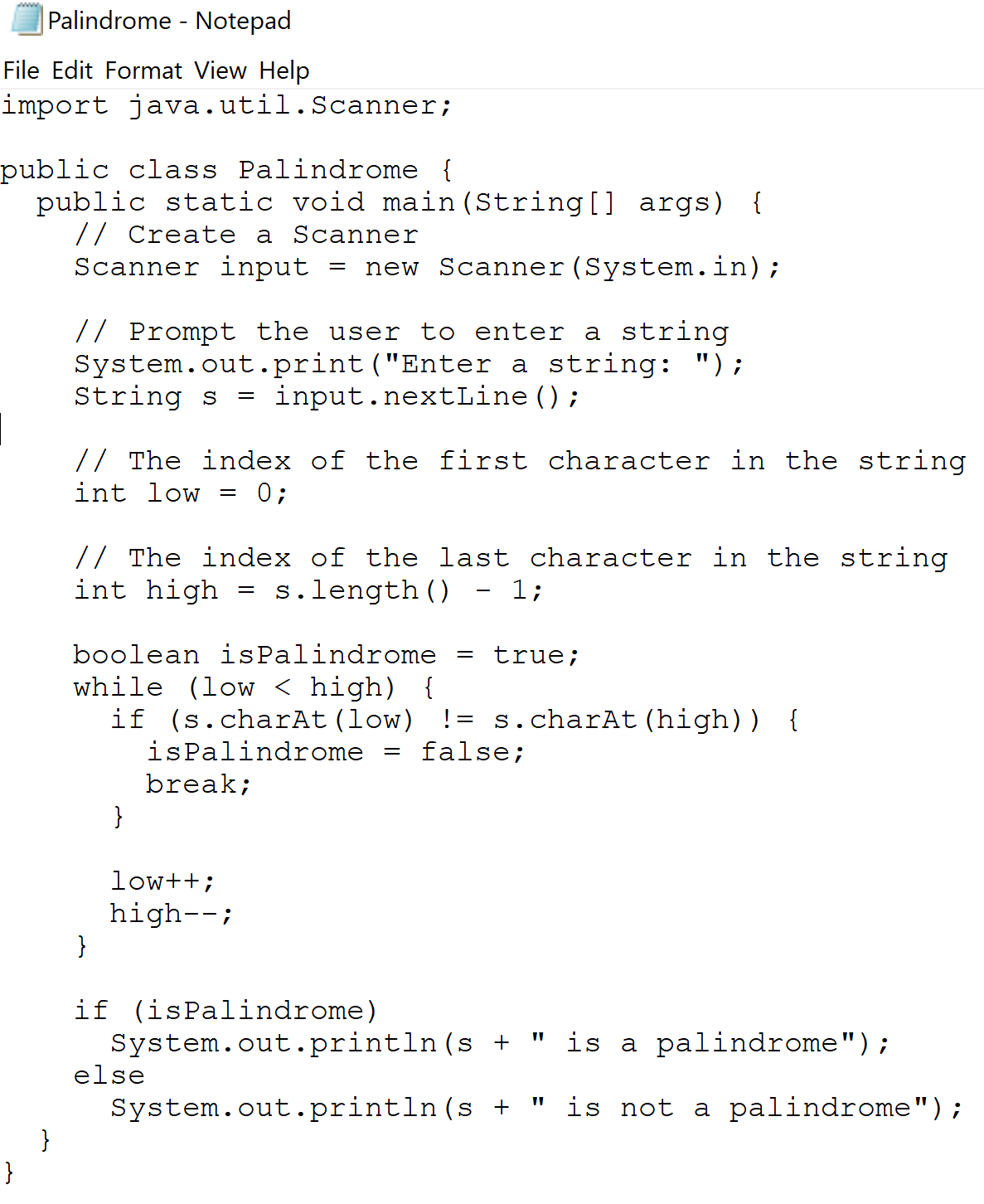
# 6) Multiplication Table.



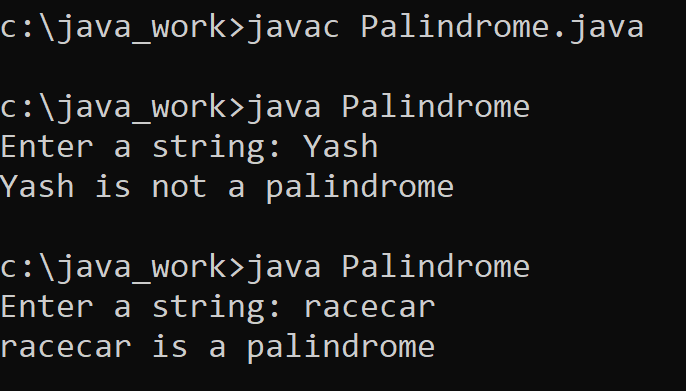
Result:



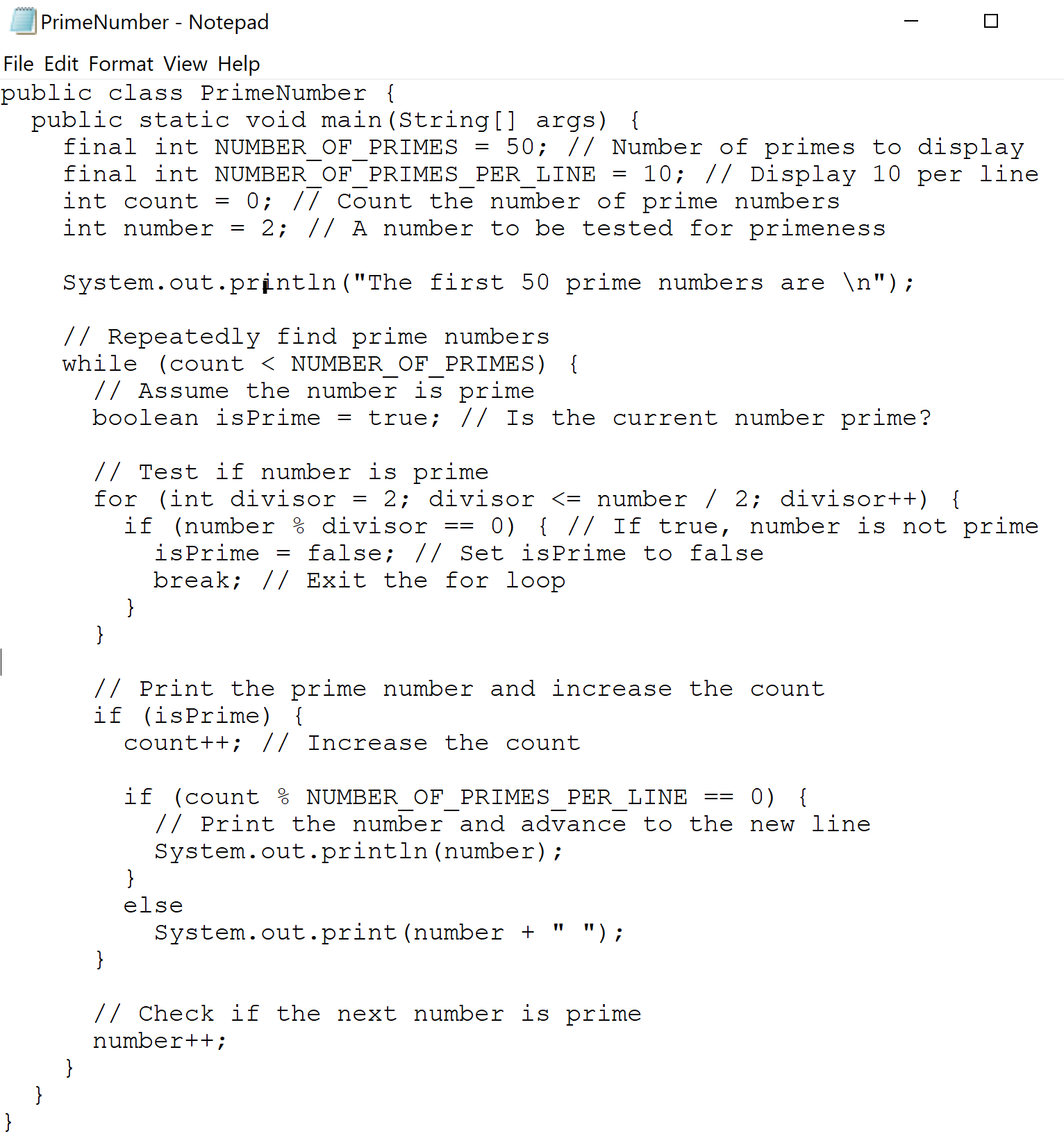
# 7) Palindrome.



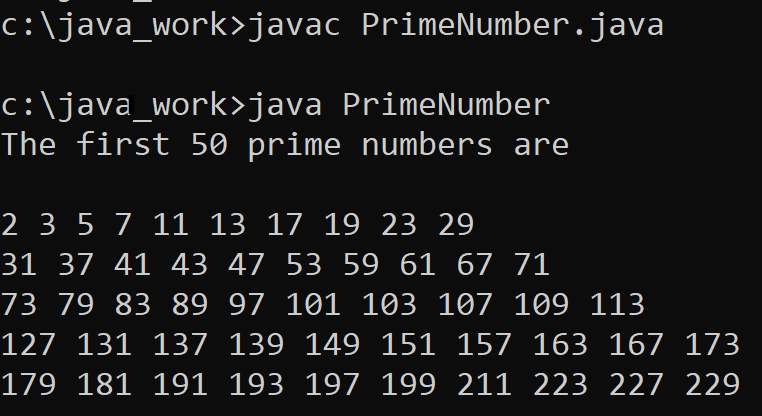
Result:



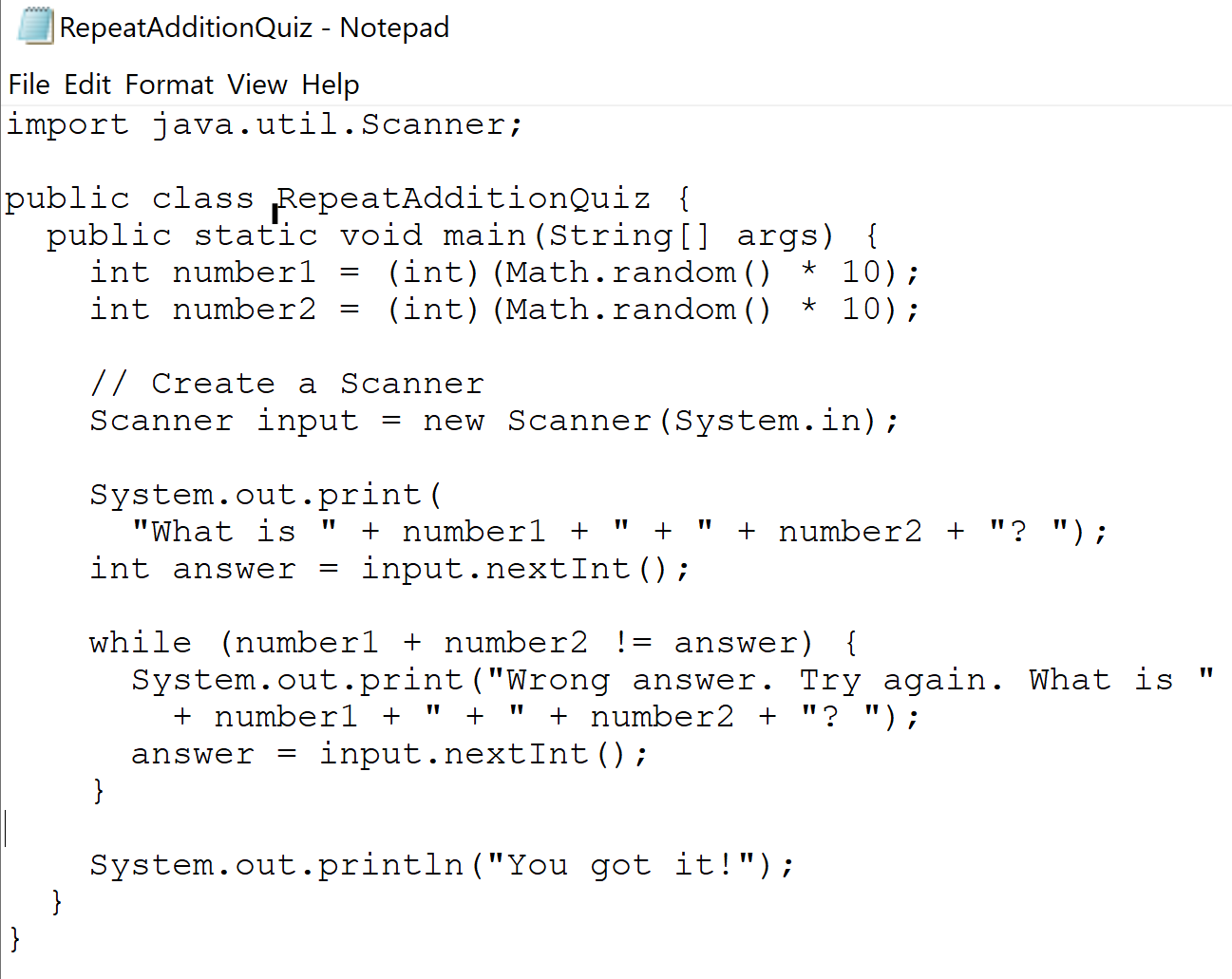
# 8) Prime Number



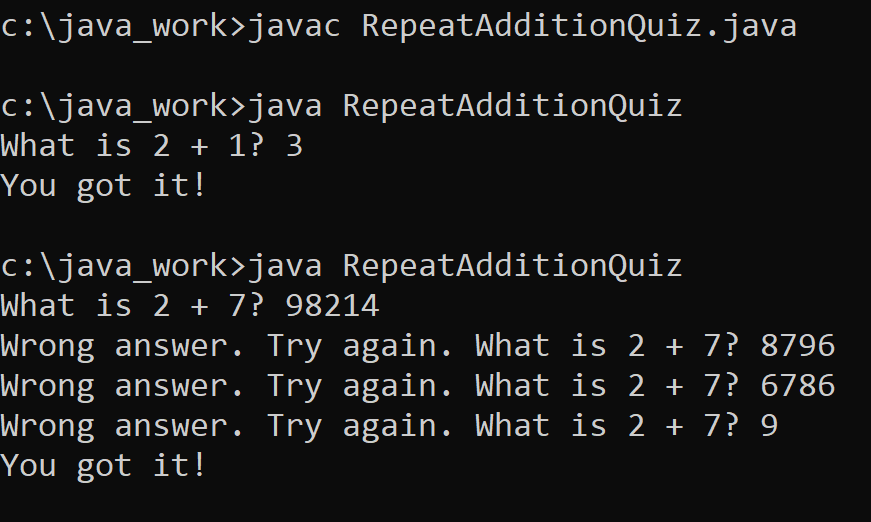
Result:



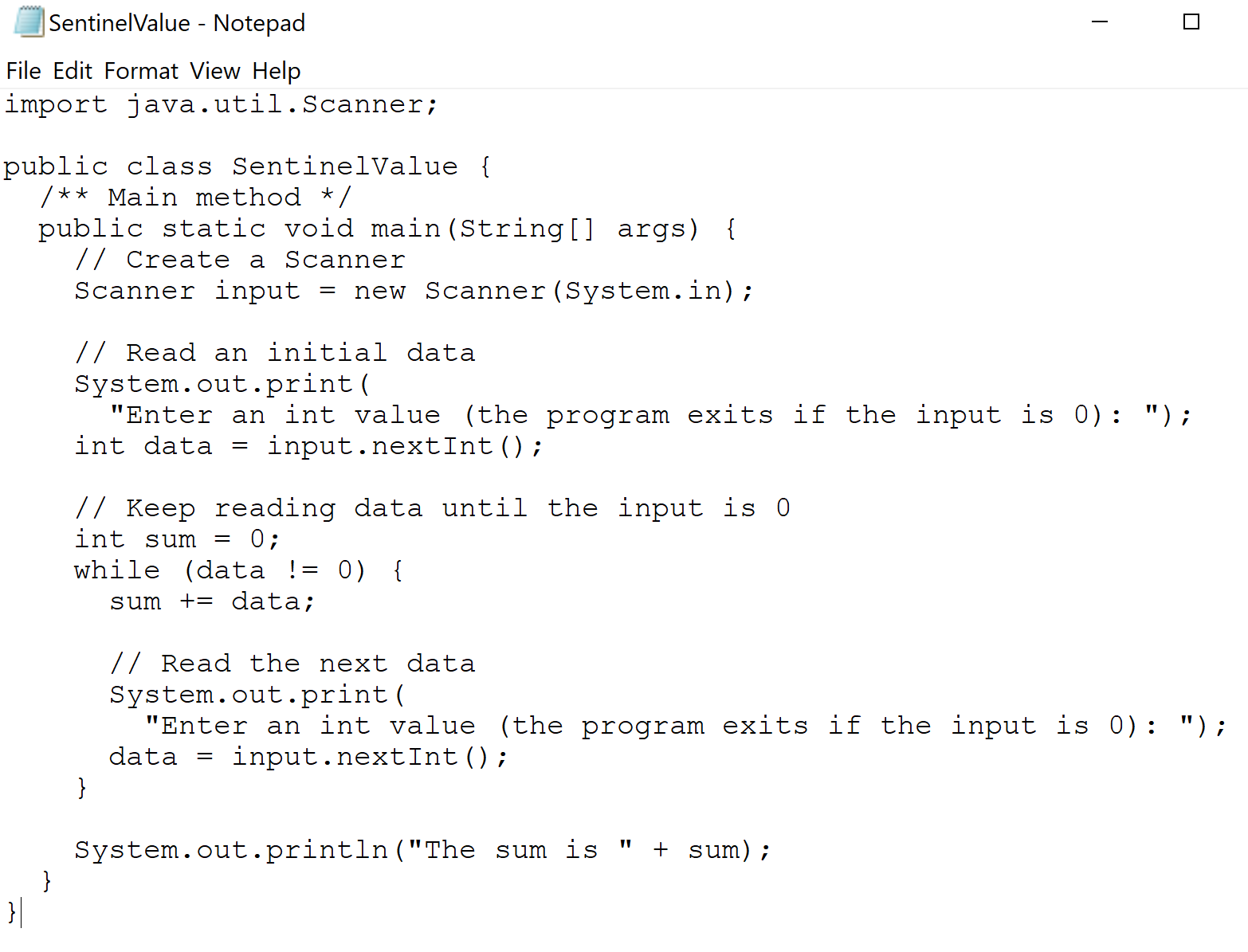
# 9) Repeat Addition Quiz.



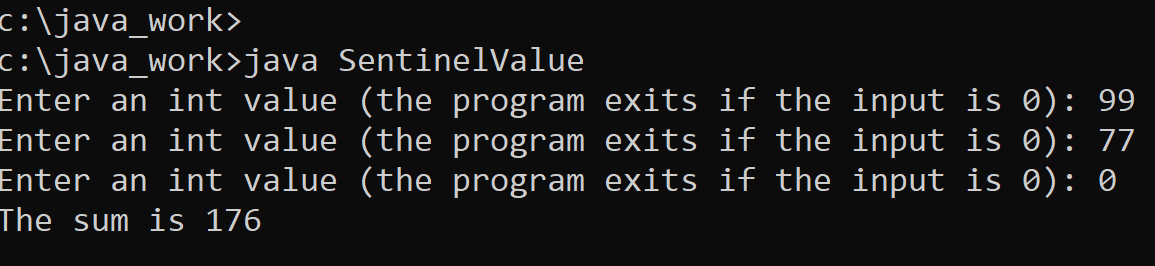
Result:



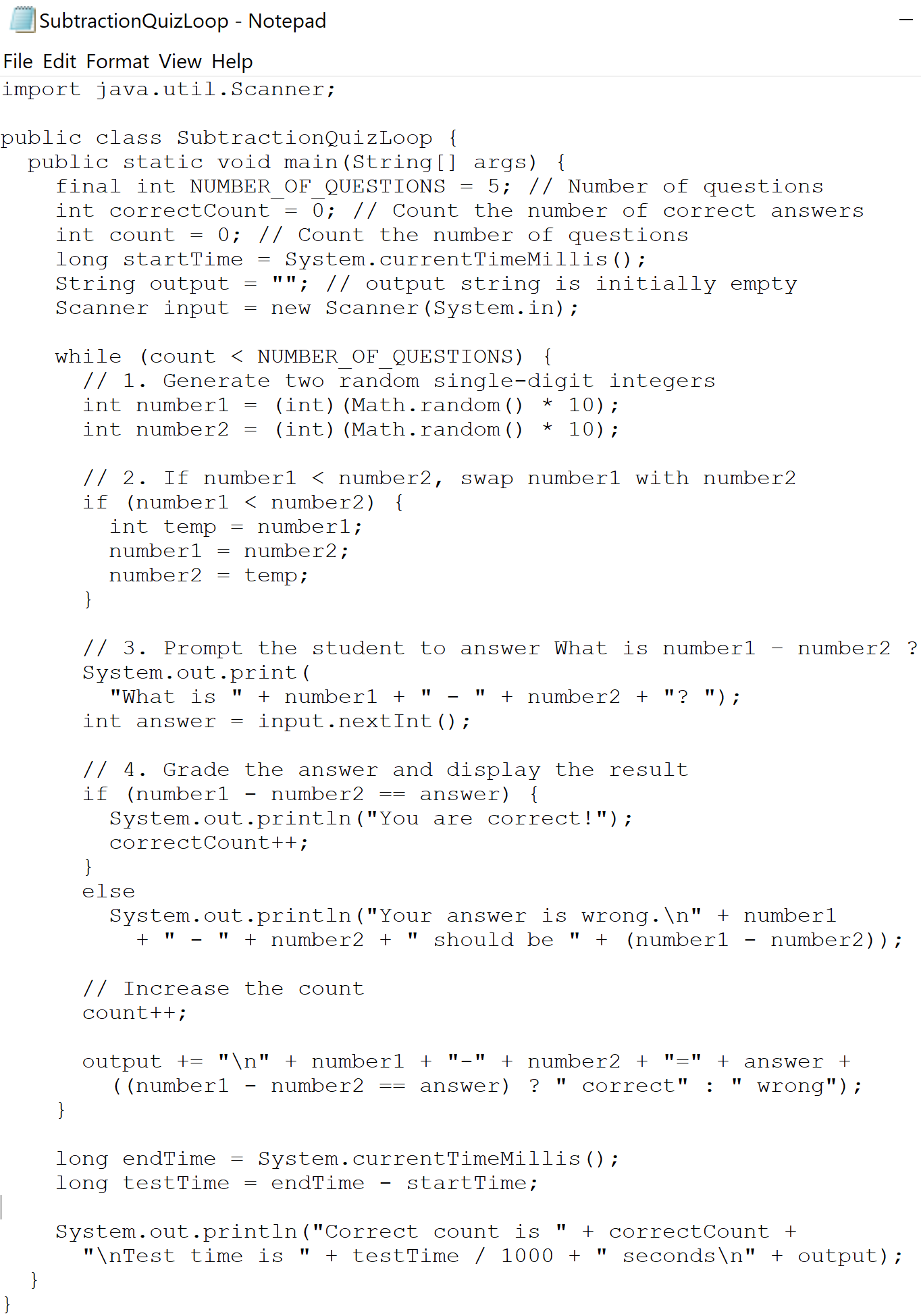
10) Sentinel Value.



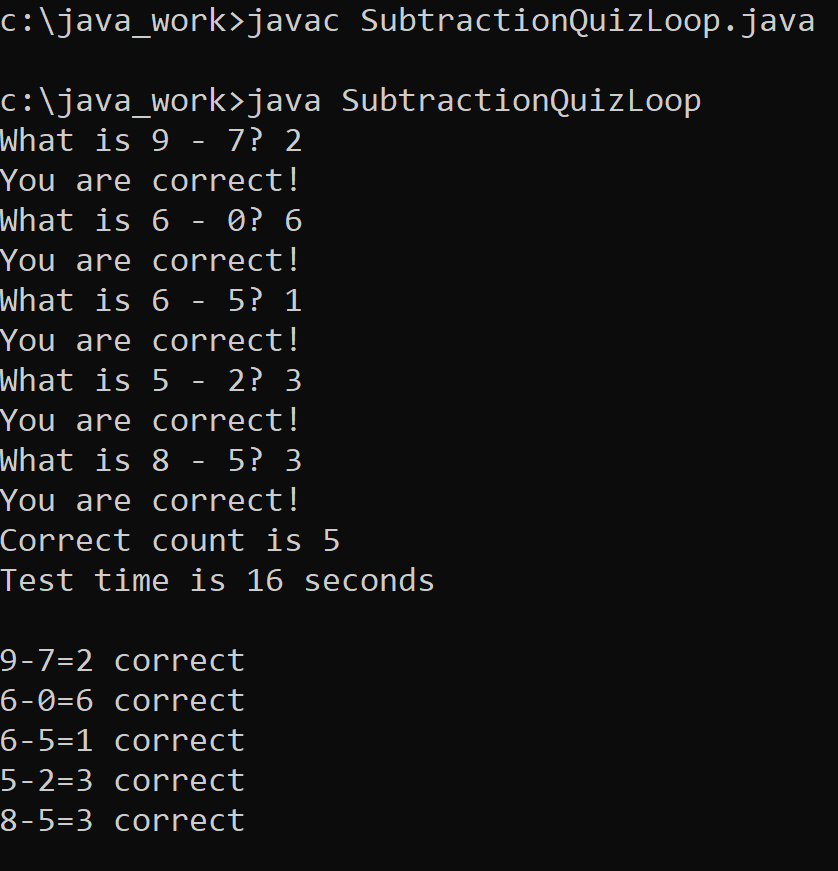
Result:



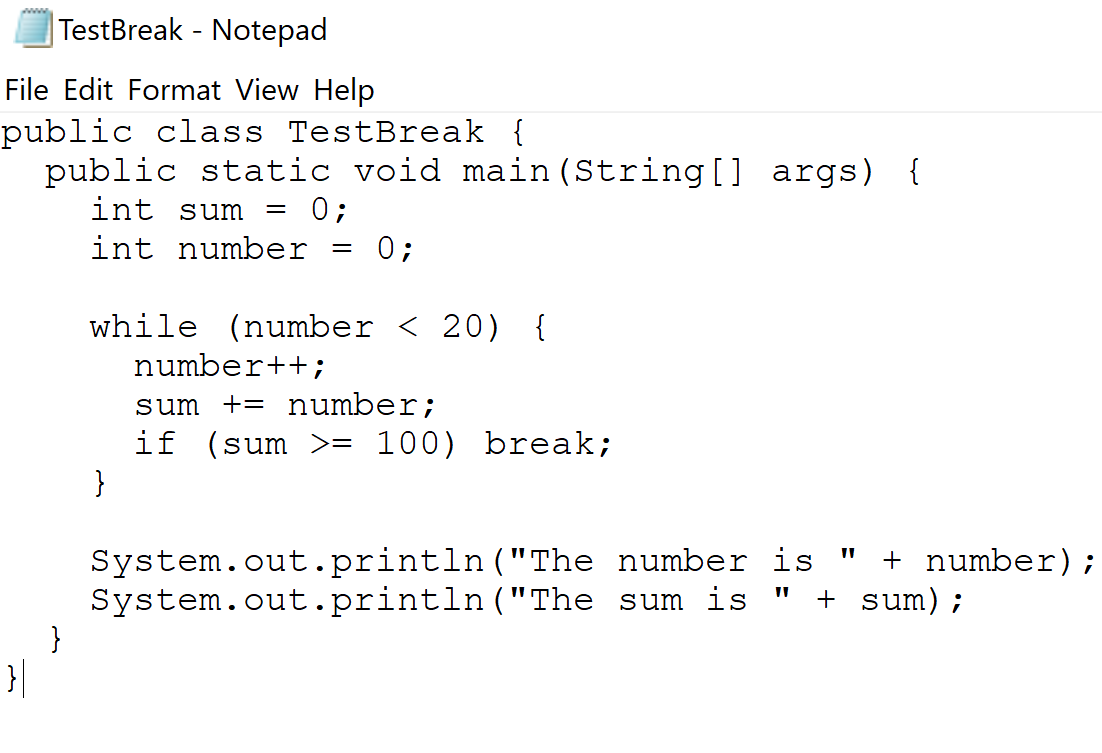
# 11) Result.



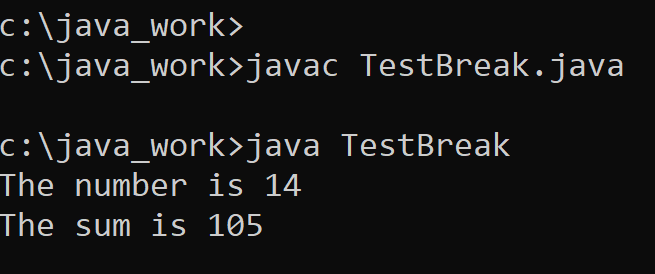
Result:



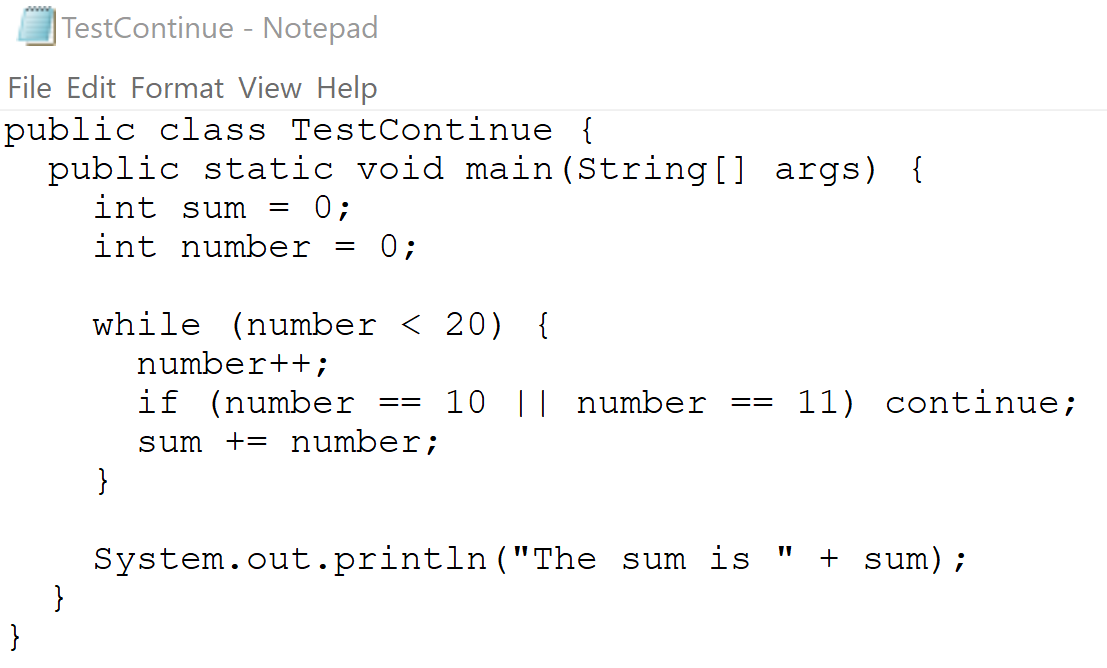
# 12) Test Break.



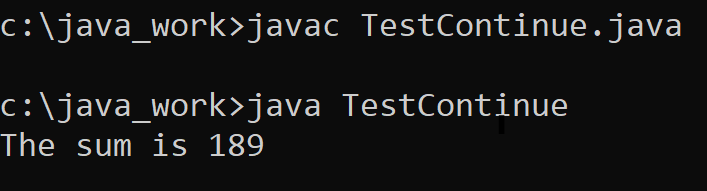
Result:



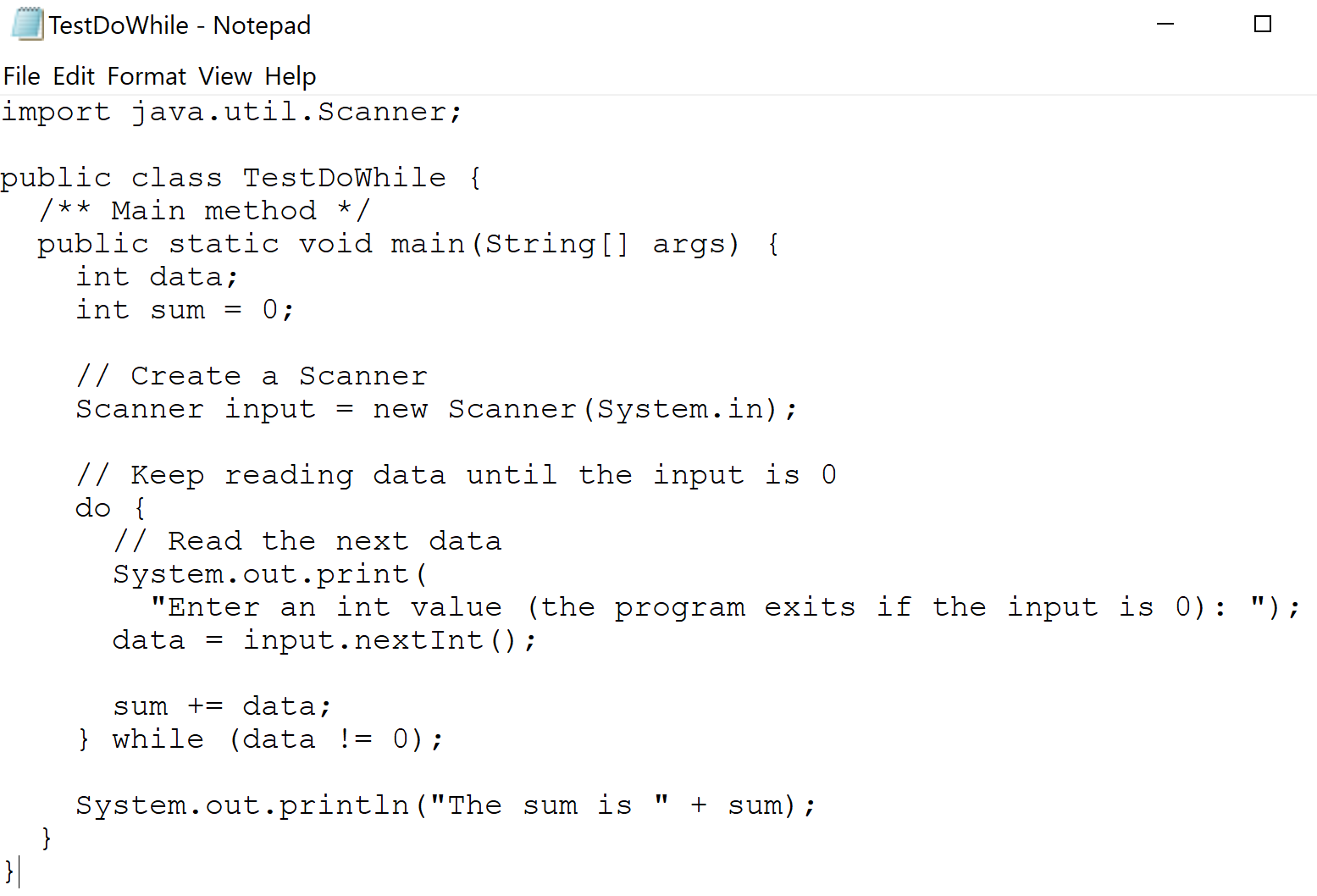
# 13) Test Continue.



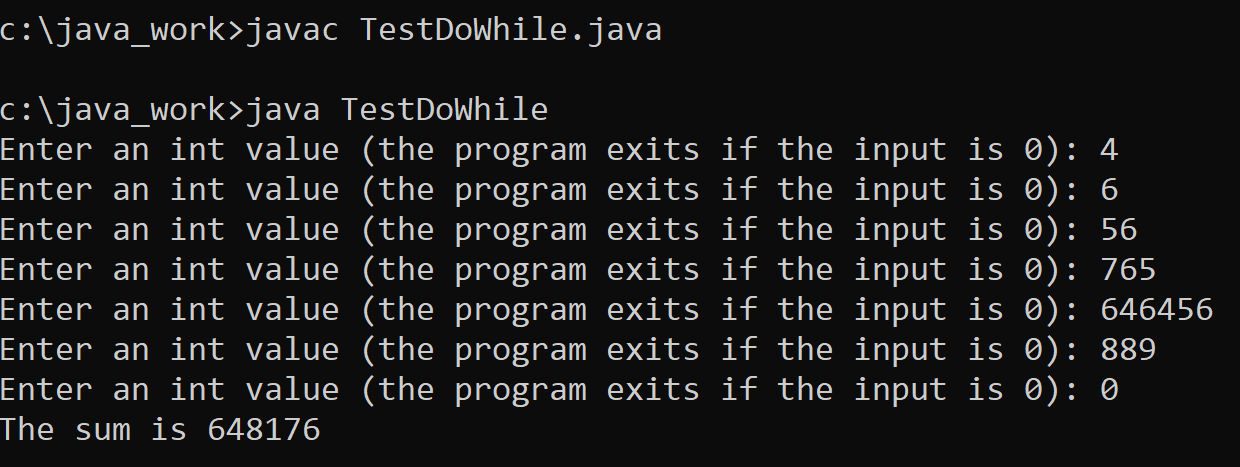
Result:



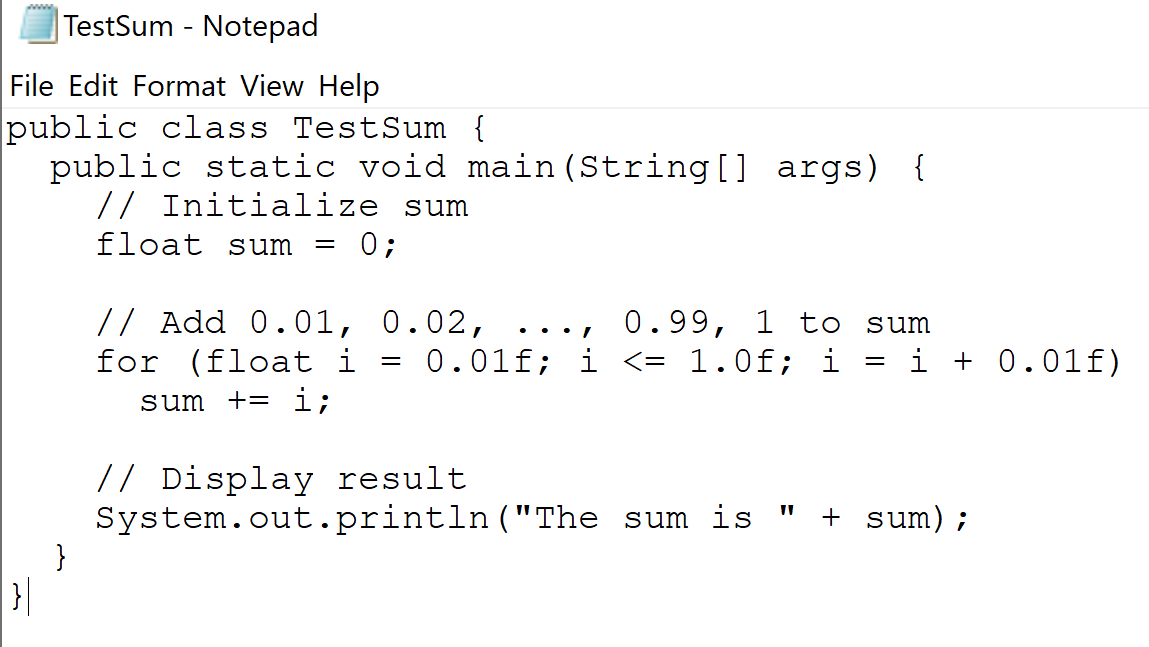
# 14) Test Do While.



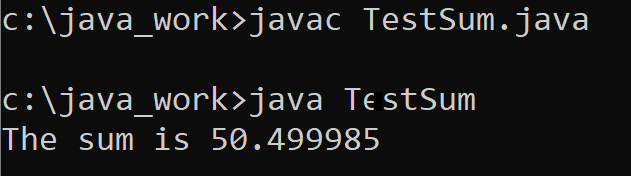
Result:



# 15) Test Sum.



Result:



Work from [Index of /02\_725/03\_ObjctClassPackge/Homework\_book/chapter6 (tfbor.com)](http://tfbor.com/02_725/03_ObjctClassPackge/Homework_book/chapter6/)

Chapter 6

(16)