NYIT

**Fall 2021**

Homework No: 05

Title: Abstract Classes

**Name:** Patade, Yash Arun

**Class ID#:**  11

**School ID#:** 1284979

**Course:** Java Networking

**Course ID:** CSCI 725

**Date:** 10/22/2021

Contents

[1) Classes 6](#_Toc87644093)

[2) Abstract Classes – OOP-SE Tool 6](#_Toc87644094)

[3) Abstract Class Example 6](#_Toc87644095)

[4) Abstract Class Example - 2 7](#_Toc87644096)

[5) Example: Abstract Classes. 8](#_Toc87644097)

[• The java.io.\* package includes several abstract classes, including these four: 8](#_Toc87644098)

[6) Defining Abstract Class. 8](#_Toc87644099)

[ An abstract class is a class that provides common behavior across a set of subclasses, but is not itself designed to have instances of its own. 8](#_Toc87644100)

[7) Rules on Abstract Class. 9](#_Toc87644101)

[ An Abstract Class cannot be instantiated 9](#_Toc87644102)

[8) Interface. 9](#_Toc87644103)

[ A class like abstract entity without any details of method implementation. 9](#_Toc87644104)

[9) Defining Interface 9](#_Toc87644105)

[An Interface defines a contract by specifying a set of method prototypes for which each class that implements it must adhere 9](#_Toc87644106)

[10) Defining Interface 10](#_Toc87644107)

[ Java interface is another source code design tool that specifies what MUST BE EXPOSED AS PUBLIC in any sub class that inherits/implements given interface. 10](#_Toc87644108)

[11) Example: Wrapper Classes 11](#_Toc87644109)

[12) Implementing Interface. 12](#_Toc87644110)

[13) Question: Interface & Abstract Class 12](#_Toc87644111)

[ Which source code entity is more dangerous to modify? 12](#_Toc87644112)

[ Which source code entity is useful when multiple inheritance is needed? 12](#_Toc87644113)

[ Which source code entity is 100% empty, has no code implementation at all? 12](#_Toc87644114)

[14) The Beginning Idea of the Interface. 13](#_Toc87644115)

[ Before writing a class definition, determine the interface 13](#_Toc87644116)

[ Similarly, if defining data structures, we should first determine the interface 13](#_Toc87644117)

[15) Data Abstraction. 13](#_Toc87644118)

[• Data (And code too) are hidden behind the interface. 13](#_Toc87644119)

[16) Interface Syntax. 14](#_Toc87644120)

[• An interface includes a collection of method declarations plus a list of constant. 14](#_Toc87644121)

[17) Example: Interface Generator 14](#_Toc87644122)

[18) Multiple Inheritance. 14](#_Toc87644123)

[• A class often needs to behave like more than one superclass. 14](#_Toc87644124)

[19) Interfaces. 15](#_Toc87644125)

[• Similar to an abstract class. 15](#_Toc87644126)

[• Interfaces provide the same functionality as multiple inheritance. 15](#_Toc87644127)

[20) Access Modifiers-Interface. 15](#_Toc87644128)

[5C) Exceptions. 16](#_Toc87644129)

[21) SimpleThread. 16](#_Toc87644130)

[22) Class TExp. 17](#_Toc87644131)

[23) Counter Extends java.applet.Applet 18](#_Toc87644132)

[24) TDriver. 19](#_Toc87644133)

[Interrupts, UNIX Signals and Handlers. 20](#_Toc87644134)

[25) How does the OS communicate to an application process? 20](#_Toc87644135)

[26) How does an application process communicate to the OS? 20](#_Toc87644136)

[27) Interrupt and jump to kernel ISR code. 20](#_Toc87644137)

[28) Interrupt: The Basic Mechanism. 21](#_Toc87644138)

[29) Types of Interrupts. 21](#_Toc87644139)

[30) Classes of Interrupts. 21](#_Toc87644140)

[31) Clarifying Terminology. 22](#_Toc87644141)

[• Interrupts may happen any time during the program execution. 22](#_Toc87644142)

[32) Exception Interrupts. 22](#_Toc87644143)

[• Exception interrupts could be: 22](#_Toc87644144)

[33) Processing interrupt. 22](#_Toc87644145)

[• Interrupt event involves presentation of interrupt requesting or IRQ signal and binary interrupt identification IID of the interrupting source device or program. 22](#_Toc87644146)

[• Interrupt event involves presentation of interrupt requesting or IRQ signal and binary interrupt identification IID of the interrupting source device or program. 23](#_Toc87644147)

[34) Interrupt priority encoder. 23](#_Toc87644148)

[• Interrupt priority encoder compares interrupt ID number with the interrupt priority level number (interrupt mask) of the currently executing program in the PSW register and issues acknowledgment ACK or denial NACK signal 23](#_Toc87644149)

[35) Interrupt priority vector address. 23](#_Toc87644150)

[36) What is an Exception? 24](#_Toc87644151)

[• An event during program execution that prevents the program from continuing normally. 24](#_Toc87644152)

[37) UNIX/Linux Signal Concepts. 24](#_Toc87644153)

[• Signals are software interrupts initiated by the OS kernel. 24](#_Toc87644154)

[38) Signal Concepts. 25](#_Toc87644155)

[• Signals are software interrupts initiated by the OS kernel. 25](#_Toc87644156)

[Java Exceptions. 26](#_Toc87644157)

[39) Example: A Little Demo. 26](#_Toc87644158)

[40) Test.java extension. 27](#_Toc87644159)

[41) Using Objects To Represent Errors. 27](#_Toc87644160)

[• Java uses Exception objects to represent errors that happen during the exception execution of your program. 27](#_Toc87644161)

[42) Exception Delivery. 28](#_Toc87644162)

[43) What To Do in Exception Handler. 28](#_Toc87644163)

[• The job of the exception handler is to put the application back into a stable state. 28](#_Toc87644164)

[44) General Type of Exception. 28](#_Toc87644165)

[• There are many kinds of exceptions that can all be represented by one class, 28](#_Toc87644166)

[Exception class. 28](#_Toc87644167)

[45) Example: Specific Types of Exceptions. 28](#_Toc87644168)

[• There are many kinds of exceptions. 28](#_Toc87644169)

[46) When is a Throws Clause Needed? 29](#_Toc87644170)

[- Method code uses throw exception object instruction. 29](#_Toc87644171)

[47) When is Throws Clause not Needed? 29](#_Toc87644172)

[48) Class Throwable. 29](#_Toc87644173)

[49) Throwable Class. 30](#_Toc87644174)

[50) Java Exception Hierarchy. 30](#_Toc87644175)

[• Members of the Exception branch of the tree are thrown to indicate the abnormal conditions that can usually be handled by the application program. 30](#_Toc87644176)

[• Hierarchy tree of the Throwable class involves more than 100 direct or indirect subclasses and can be shown only partially. 31](#_Toc87644177)

[51) Types (Sub Types) of Exceptions. 31](#_Toc87644178)

[• All exceptions in Java are objects of Throwable class. 31](#_Toc87644179)

[52) Unchecked Exceptions. 32](#_Toc87644180)

[• Class **Error** & **RunTimeException**. 32](#_Toc87644181)

[53) Handling Exceptions in Java. 33](#_Toc87644182)

[• Exception mechanism is built around the throw-and-catch paradigm. 33](#_Toc87644183)

[54) System and Application Exceptions. 33](#_Toc87644184)

[55) System Exceptions. 33](#_Toc87644185)

[56) Application Exceptions. 34](#_Toc87644186)

[• Application exceptions are not catastrophic in nature. 34](#_Toc87644187)

[57) Example: Uncaught Exception. 34](#_Toc87644188)

[58) StackDemo. 36](#_Toc87644189)

[References 37](#_Toc87644190)

# 1) Classes

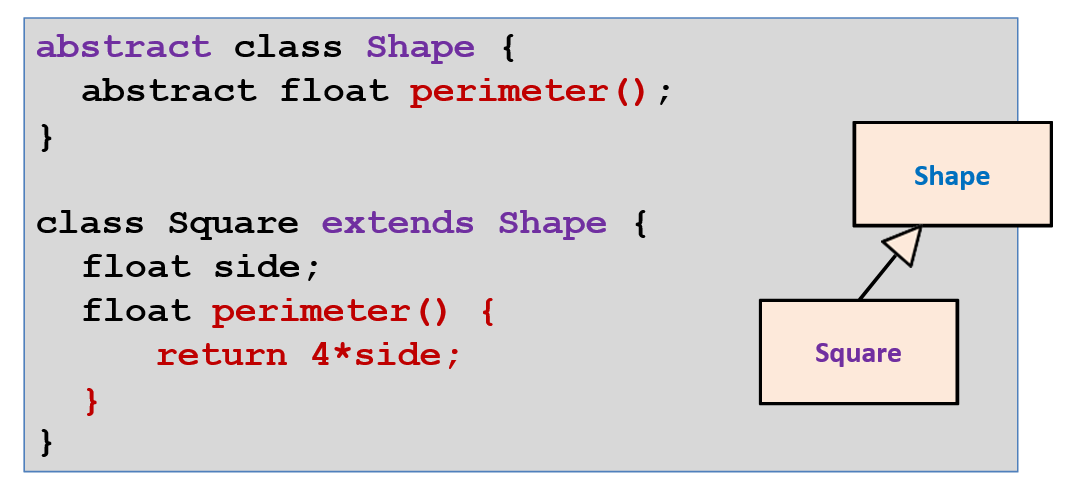
* Class may be:
  + Concrete – complete to be loaded and used, e.g., to create objects using new operator.
  + Abstract – incomplete class that has to be completed through inheritance subclasses.
    - One cannot declare an object of an incomplete abstract class.

# 2) Abstract Classes – OOP-SE Tool

* An abstract class is
  + A model or class-design-pattern for subclasses,
  + A plan how subclasses should look.
  + Incomplete class code.
* An abstract class may declare methods which are themselves abstract-they have no bodies.
* Each subclass of an abstract class which is not itself abstract must override the abstract methods.

# 3) Abstract Class Example

* Abstract class is OOP code engineering/design blueprint that MUST BE IMPLEMENTED by the programmer.

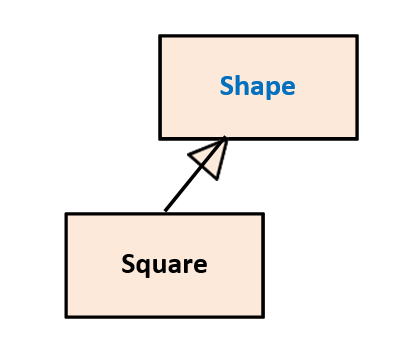


# 4) Abstract Class Example - 2

* You can write:

**Square s = new Square();**

**Shape s = new Square();**



# 5) Example: Abstract Classes.

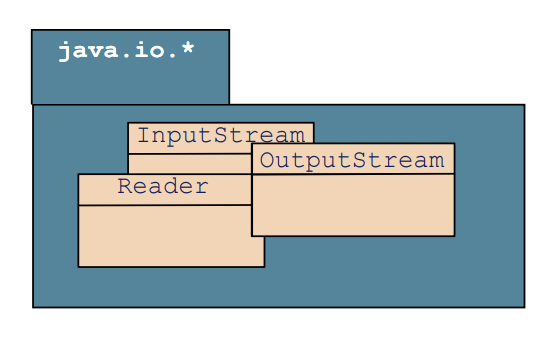
## • The java.io.\* package includes several abstract classes, including these four:

– InputStream

– OutputStream

– Reader

– Write



# 6) Defining Abstract Class.

## An abstract class is a class that provides common behavior across a set of subclasses, but is not itself designed to have instances of its own.

* A class that dictates certain behavior but allows its subclasses to provide implementation
* A class designed only as a parent from which sub-classes may be derived, but which is not itself suitable for instantiation
* A class often used to "abstract out" incomplete sets of features which may then be shared by a group of sibling sub- classes which add different variations of the missing pieces.

# 7) Rules on Abstract Class.

## An Abstract Class cannot be instantiated

* An abstract class can/SHOULD be extended
* An abstract class can have any number of abstract methods or none at all
* A class with at least one abstract method must be declared an abstract class
* A subclass can provide partial (is abstract too) or full implementations of the inherited abstract methods
* A class that has at least one abstract method, whether declared or inherited from an abstract class, must be declared abstract.

# 8) Interface.

## A class like abstract entity without any details of method implementation.

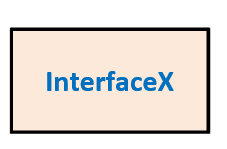
More abstract than:

* + - a class or
    - an abstract class.

# 9) Defining Interface

## An Interface defines a contract by specifying a set of method prototypes for which each class that implements it must adhere

– An interface is 100% abstract class

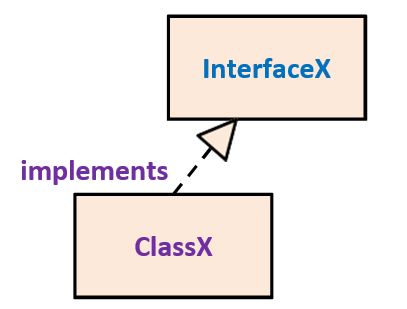


# 10) Defining Interface

## Java interface is another source code design tool that specifies what MUST BE EXPOSED AS PUBLIC in any sub class that inherits/implements given interface.

* Interface implementation solves the problem of multiple inheritance, when designers must have the subclass that inherits from two super classes.
* In Java, the second, third, etc. inheritance must be from the 100% abstract (Empty code - Must be implemented) class.

– 100% abstract class in Java is known as interface.

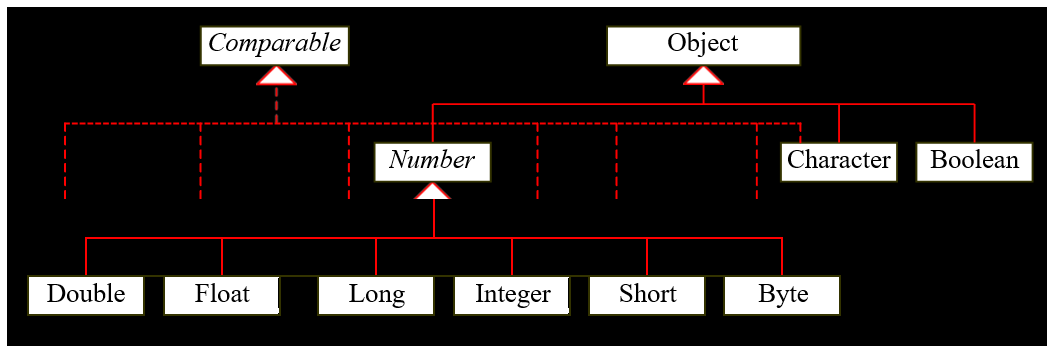


* An interface provides only a form for a class but no implementation
* An interface defines *what* a class can do but not *how* the class will do it, (How – must be programmed/implemented in source code)

# 11) Example: Wrapper Classes

* Boolean
* Character
* Short
* Byte
* Integer
* Long
* Float
* Double

1. The wrapper classes do not have no-arg constructors.
2. The instances of all wrapper classes are immutable, i.e., their internal values cannot be changed once the objects are created.



# 12) Implementing Interface.

* Implementing an interface means providing implementations for its methods
* Interfaces are implemented using the **implements**
* keyword
* Rules on implementing the interface methods
* Must have the same method signature and return type
* Cannot narrow the method accessibility
* Cannot specify broader checked exceptions
* Interface variables are implicitly **public final static**
* Interface methods are implicitly **public abstract**

# 13) Question: Interface & Abstract Class

## Which source code entity is more dangerous to modify?

* 1. Interface, or
  2. Abstract class

Answer) Interface.

## Which source code entity is useful when multiple inheritance is needed?

* 1. Interface, or
  2. Abstract class

Answer) Interface.

## Which source code entity is 100% empty, has no code implementation at all?

* 1. Interface, or
  2. Abstract class

Answer) Interface.

# 14) The Beginning Idea of the Interface.

## Before writing a class definition, determine the interface

- the set of services class-object offer to clients

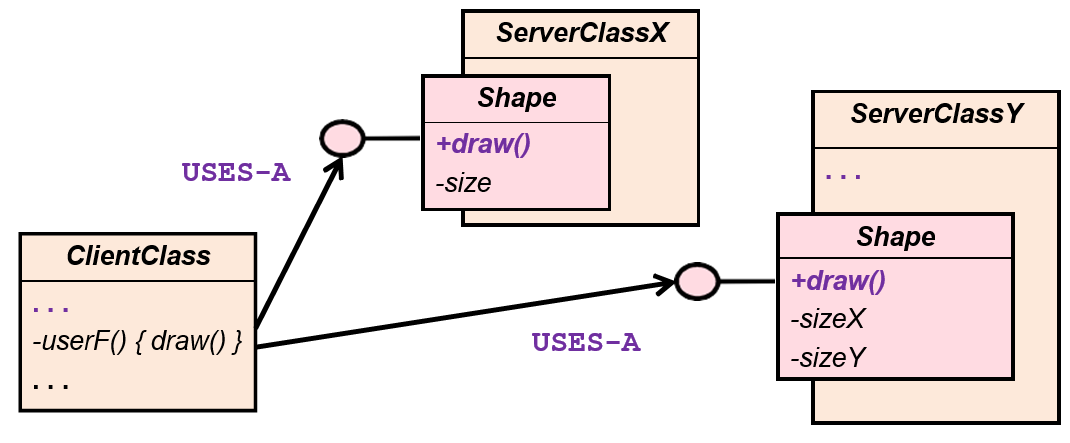
## Similarly, if defining data structures, we should first determine the interface

* stacks support a constructor, push, pop, size, isEmpty, and top
* queues offer a constructor, enqueue, dequeue, size, isEmpty and front

# 15) Data Abstraction.

## Data (And code too) are hidden behind the interface.

* From the client-class-code point of view, (who does not know the data), data could be anything.
* To client-class code, service-object private data are abstract??



# 16) Interface Syntax.

## An interface includes a collection of method declarations plus a list of constant.

* + Describes a set of methods that a class can be forced to implement.
  + An interface has no variable declarations or no method bodies.
  + An interface can be a refinement of another interface by using extend.

# 17) Example: Interface Generator

* Goal
  + Develop several implementations of an object, i.e.

*generator,* that generate different sequence of integers, such as

* + - constants - *ConstantGenerator*
    - random integers – *RandomGenerator*
    - primes – *PrimeGenerator*
* Interface Generator

# 18) Multiple Inheritance.

## A class often needs to behave like more than one superclass.

* Example: The **RandomAccessFile** class should contain all the methods of the **DataInput** class and also the **DataOutput** class.
* But in Java, each class has exactly one superclass.

# 19) Interfaces.

## Similar to an abstract class.

* Key difference: a class can implement many interfaces but it can extend only one class.
* An interface is a collection of
  + zero or more class variables (treated as final, keyword final is optional)
  + zero or more instance method names without bodies (keyword abstract is optional)
  + A class can be defined as implementing any number of interfaces. The class must provide the method implementations.

## Interfaces provide the same functionality as multiple inheritance.

* An interface cannot define.

- Class methods (because class methods are statically bound to the class they are defined in)

- Instance variables (or objects)

# 20) Access Modifiers-Interface.

* **public**

- The interface is accessible anywhere the interface name is accessible.

<no modifier>

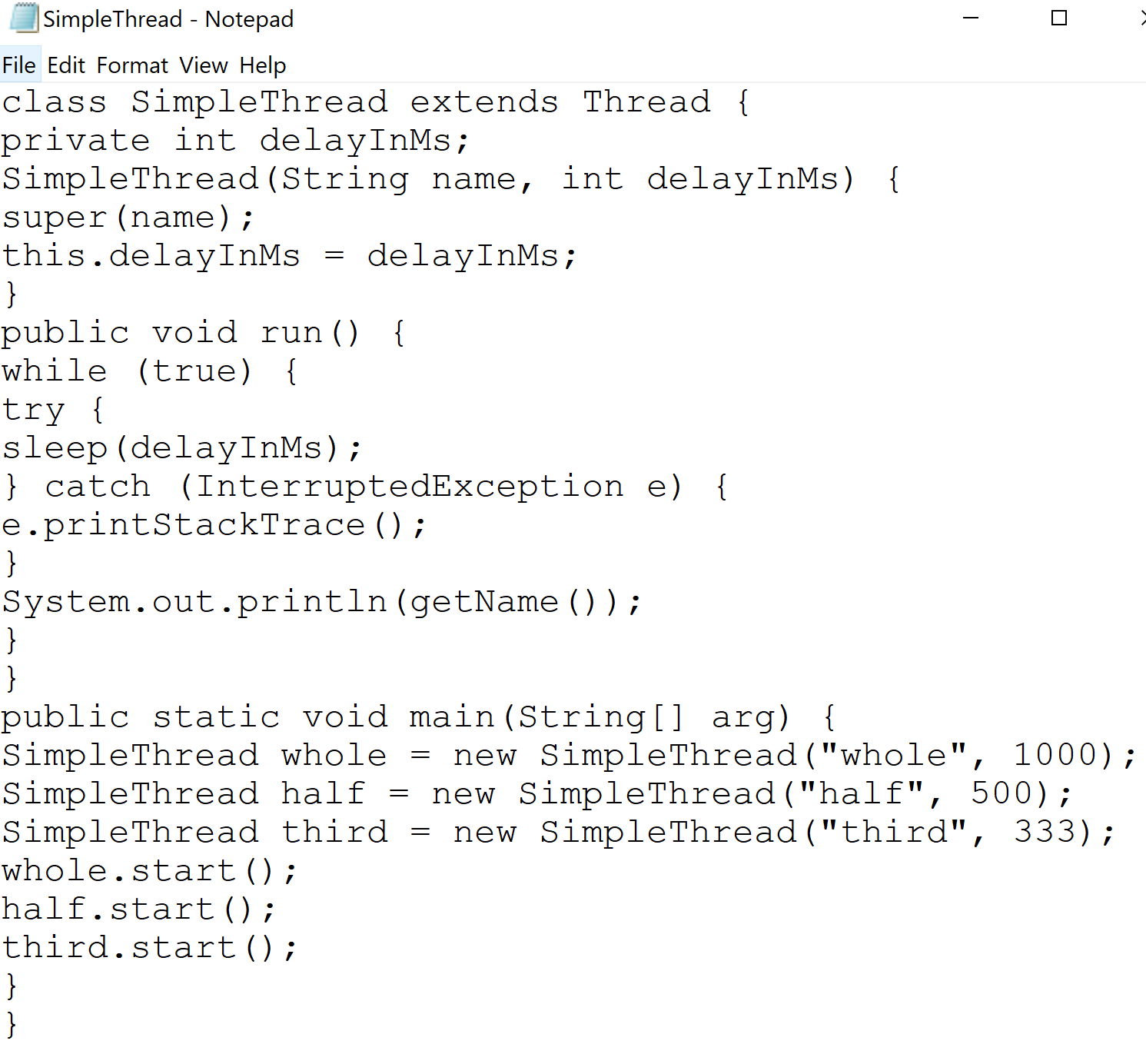
- The interface is only visible to classes within the same package.

* **abstract**

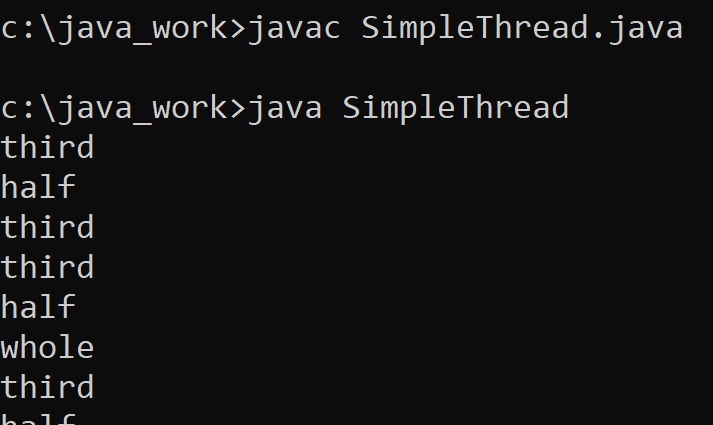
- Do not use this modifier; it is obsolete in this context.

# 5C) Exceptions.

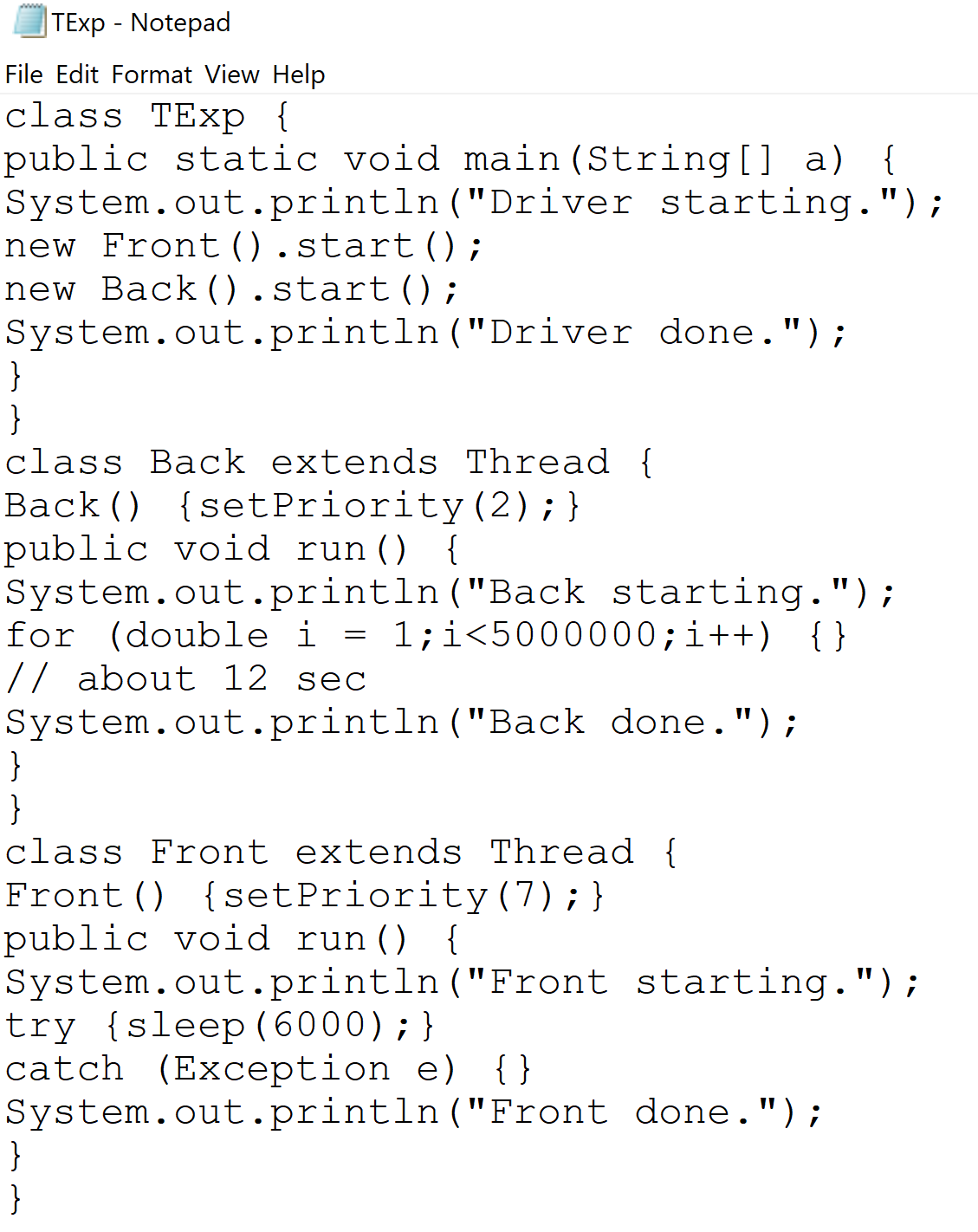
# 21) SimpleThread.



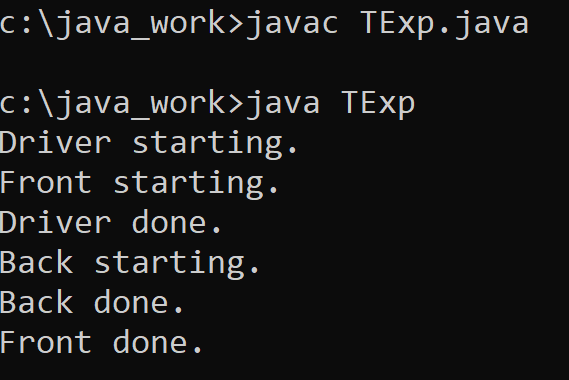
Result)



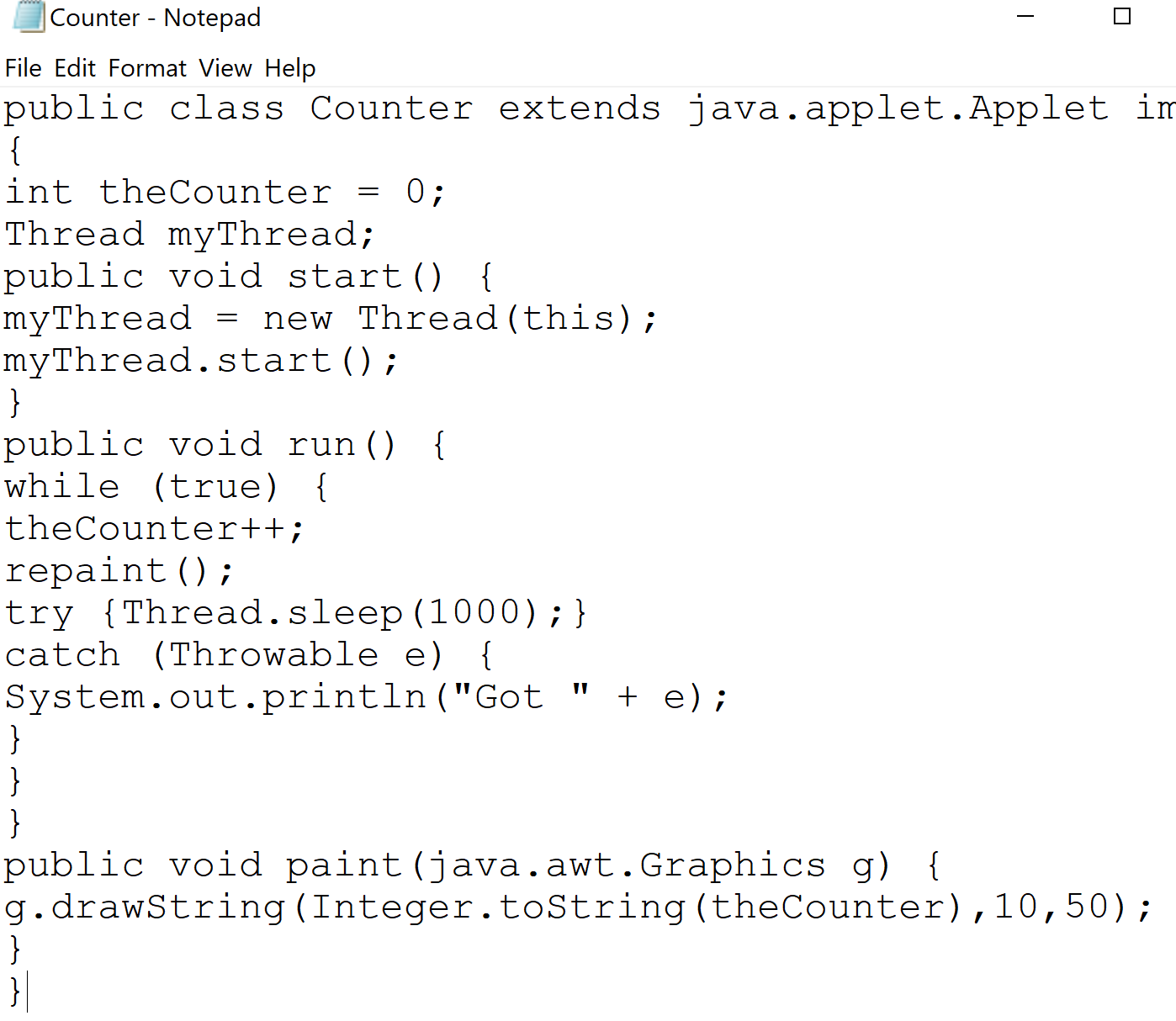
# 22) Class TExp.



Result)

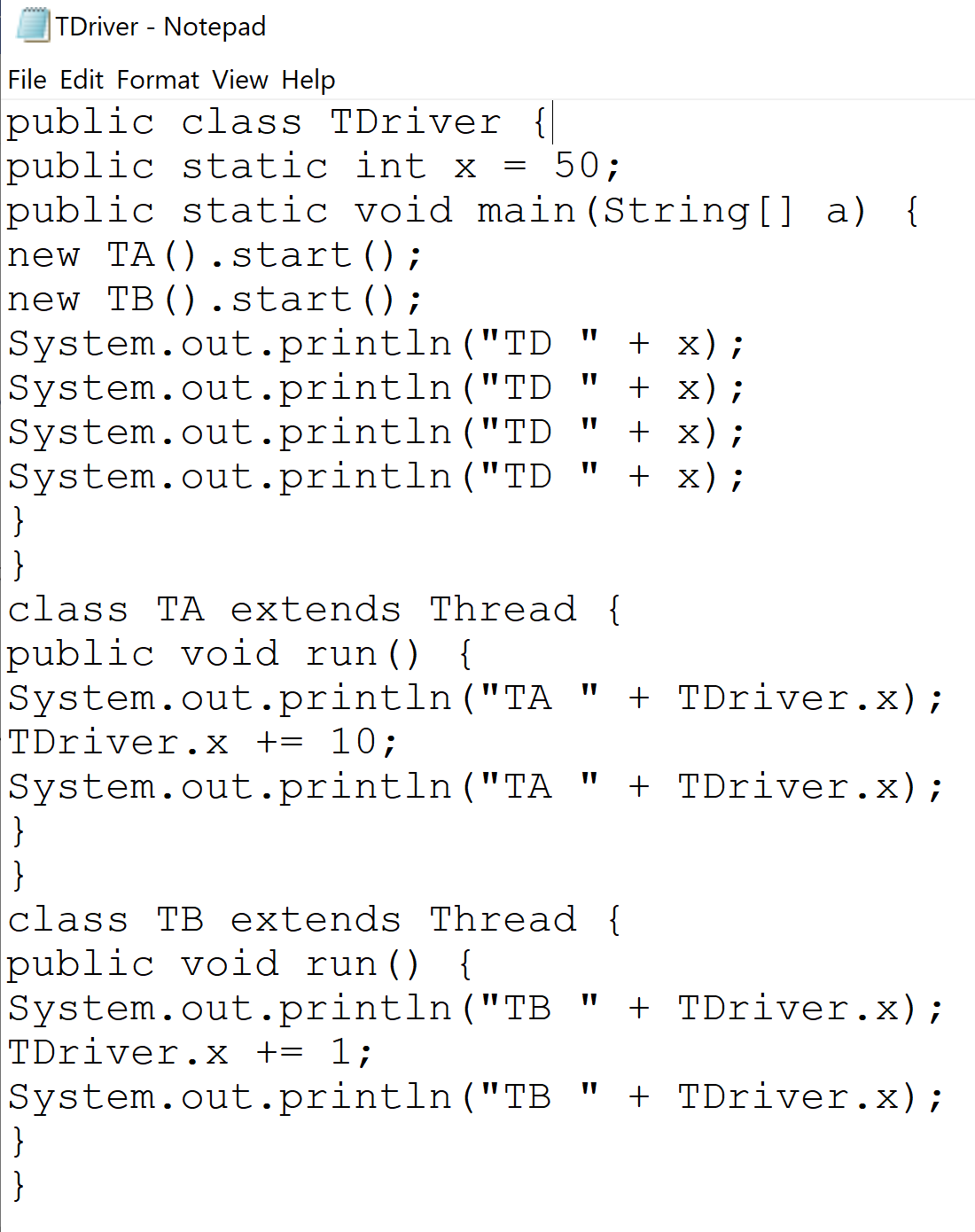


# 23) Counter Extends java.applet.Applet

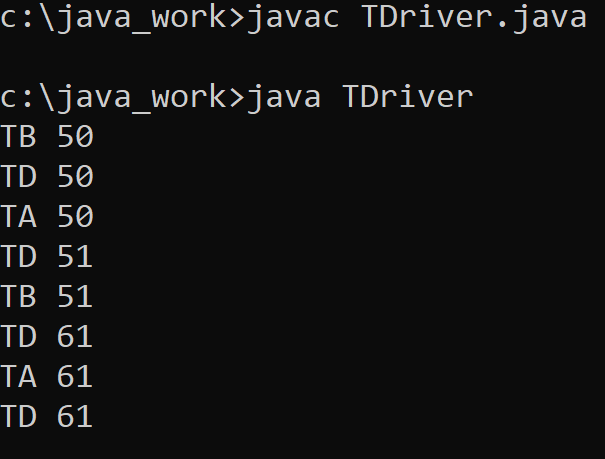


Answer) The program does not compile because this cannot be referenced in a static method.

# 24) TDriver.



Result)



# Interrupts, UNIX Signals and Handlers.

# 25) How does the OS communicate to an application process?

# 26) How does an application process communicate to the OS?

Answer)

• Interrupt: A thread flow control mechanism that enables a device or program to notify the CPU that another program should immediately run.

– Example: Device controller “raises” an interrupt by setting a signal on the interrupt-request line, and the CPU “catches” it and “dispatches” it to the interrupt handler.

# 27) Interrupt and jump to kernel ISR code.

* Any interrupt event triggers program thread jump out of code segment CSP to the kernel code segment CSK.
* Particular kernel code module that runs upon interrupt-caused jump is Interrupt Handler IH or Interrupt Service Routine, ISR.

# 28) Interrupt: The Basic Mechanism.

1. The CPU receives the interrupt signal and ID number of the source of the interrupt,

2. Control (PCr of the current program) is transferred to the OS (Jump into the Kernel’s interrupt handler),

3. Current state is saved,

4. The request is serviced (Handled),

5. Previous state of the interrupted program is restored,

6. Control is returned to the previously interrupted program.

# 29) Types of Interrupts.

• Hardware interrupts (also called external/asynchronous interrupts), are ones in which the notification originates from a hardware device such as a keyboard, mouse or system clock.

• Software interrupts include exceptions and traps.

Exceptions: triggered by an action of the process without its knowledge (division by zero, access to paged memory, etc.)

Traps: triggered by the process using system calls.

• Usually, no interrupt should be ignored by the OS.

# 30) Classes of Interrupts.

• We may classify interrupts into following three classes:

• Hardware caused interrupts,

• Exception caused interrupts, and

• Software caused interrupts

# 31) Clarifying Terminology.

## • Interrupts may happen any time during the program execution.

– Exceptions (OS) - A type of hardware interrupt generated by the CPU itself, based upon the operands of certain instructions. [http://wiki.osdev.org/Exceptions]

• Faults - Recoverable error. (Div by Zero, Page)

• Traps - Executed immediately after instruction. (Breakpoint, Intel “INTO” interrupt on overflow)

• Aborts - Severe unrecoverable error. (Machine check, auch as bad memory or internal fault)

# 32) Exception Interrupts.

## • Exception interrupts could be:

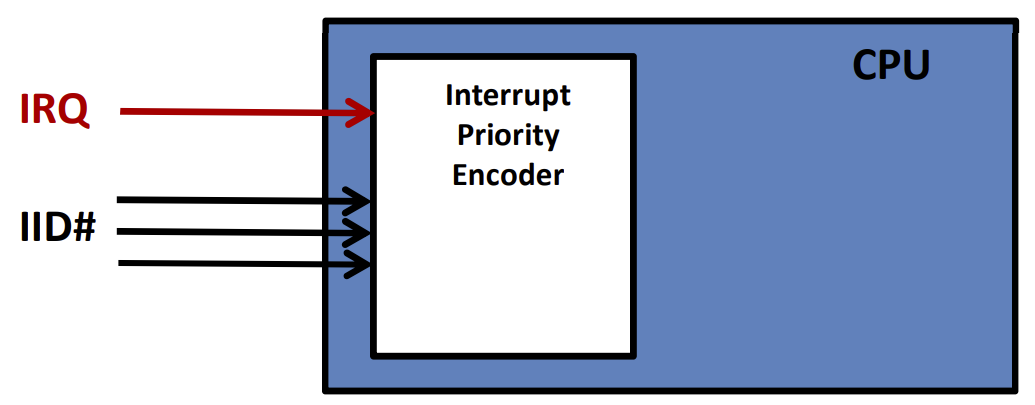
– Faults, (An error in an instruction is caught before it is executed)

– Traps, (An error in an instruction is detected after it is executed)

– Aborts, (An error in an instruction is detected, but the offending instruction nature can’t be identified).

# 33) Processing interrupt.

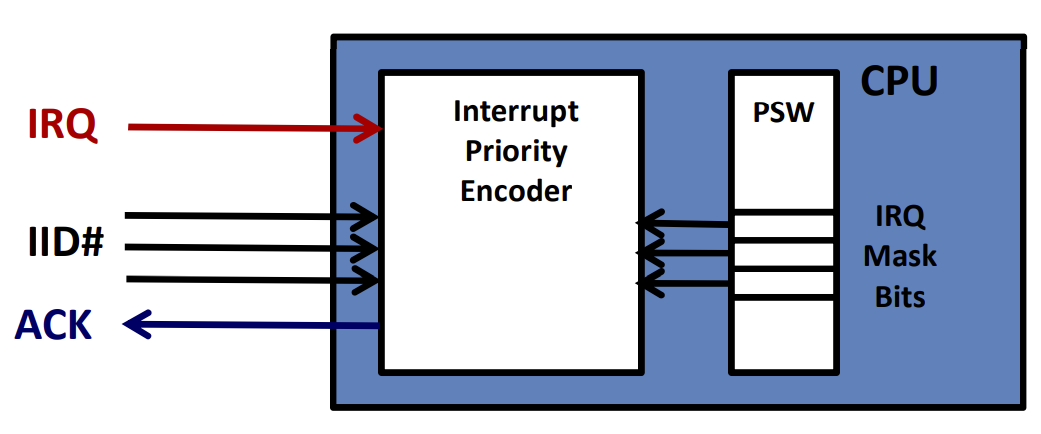
## • Interrupt event involves presentation of interrupt requesting or IRQ signal and binary interrupt identification IID of the interrupting source device or program.



## • Interrupt event involves presentation of interrupt requesting or IRQ signal and binary interrupt identification IID of the interrupting source device or program.

# 34) Interrupt priority encoder.

## • Interrupt priority encoder compares interrupt ID number with the interrupt priority level number (interrupt mask) of the currently executing program in the PSW register and issues acknowledgment ACK or denial NACK signal

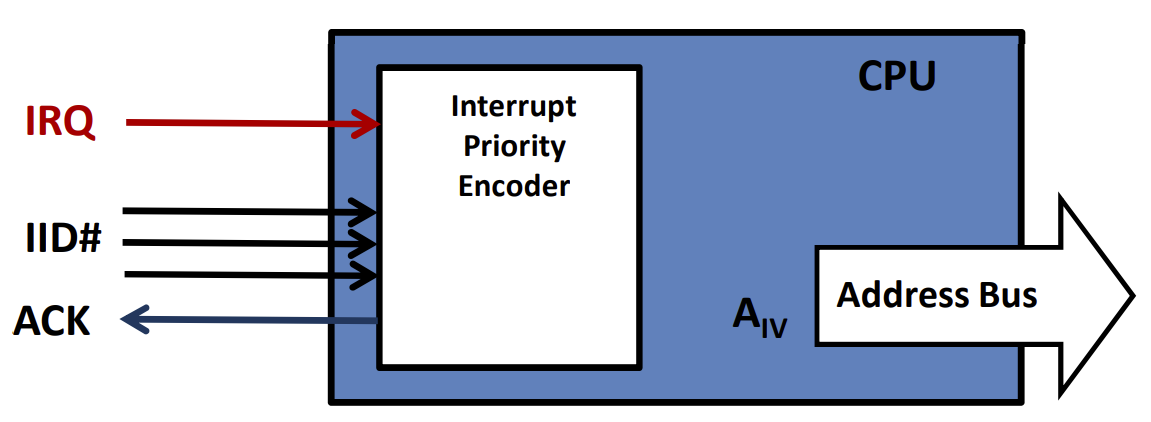


# 35) Interrupt priority vector address.

• In case of IRQ acknowledgement IID number is sued to calculate address of Interrupt Vector (IV).

For instance:

**AIV = IVA = 4 \* IID**



# 36) What is an Exception?

## • An event during program execution that prevents the program from continuing normally.

• An error condition that changes the normal flow of control in a program.

• A signal that some unexpected condition has occurred in the program.

• An unusual/unexpected abnormal thread flow that continues in the handler code.

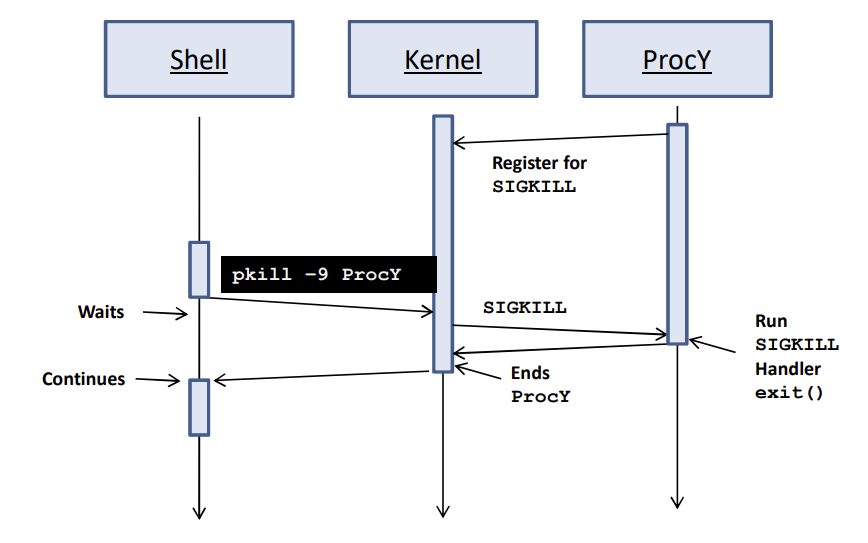
– Handler should be custom made as non-standard way/code.

# 37) UNIX/Linux Signal Concepts.

## • Signals are software interrupts initiated by the OS kernel.

– Used to handle asynchronous events.

– Command pkill may happen any time and return must wait for return.



# 38) Signal Concepts.

## • Signals are software interrupts initiated by the OS kernel.

– Used to handle asynchronous events. • Signals in unix have names and numbers.

• Use ‘man kill’ to see the types of signals.

• Each signal has a positive integer ID and standardized name.

• All the signals start with SIG.

– SIGKILL = 9

– SIGTERM = 15

– SIGCHLD = 17

– SIGINT = 2

• Signals are defined in <signal.h>

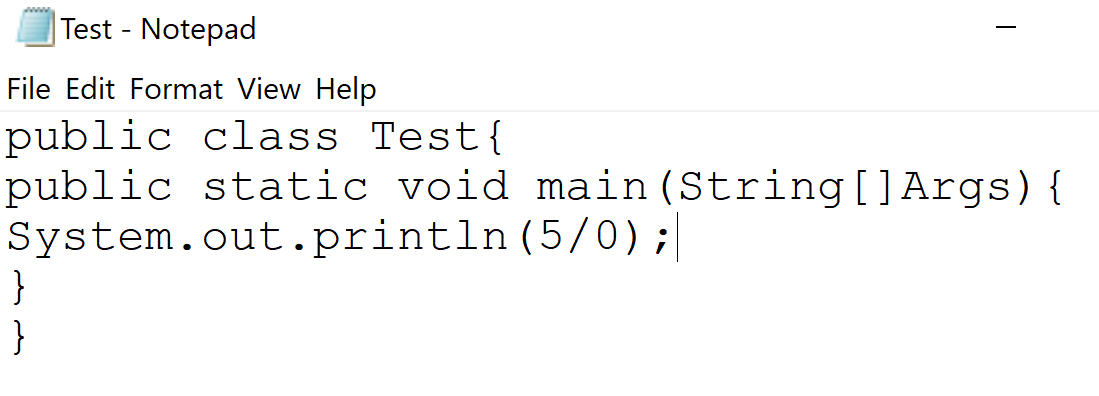
• man 7 signal for complete list of signals and their numeric values.

• kill –l for full list of signals on a system.

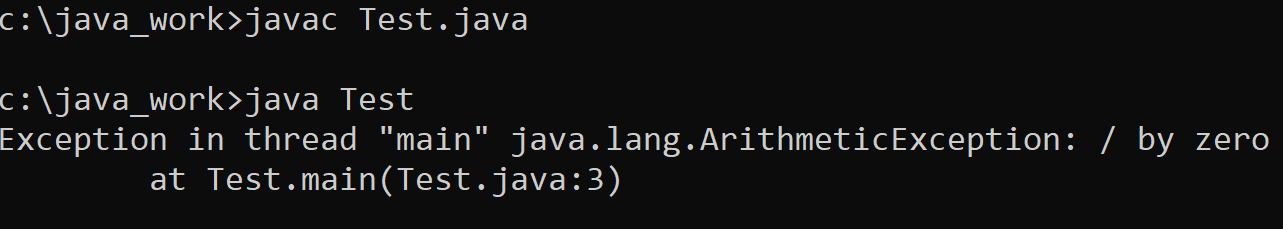
– 64 signals. The first 32 are traditional signals, the rest are for real time applications.

# Java Exceptions.

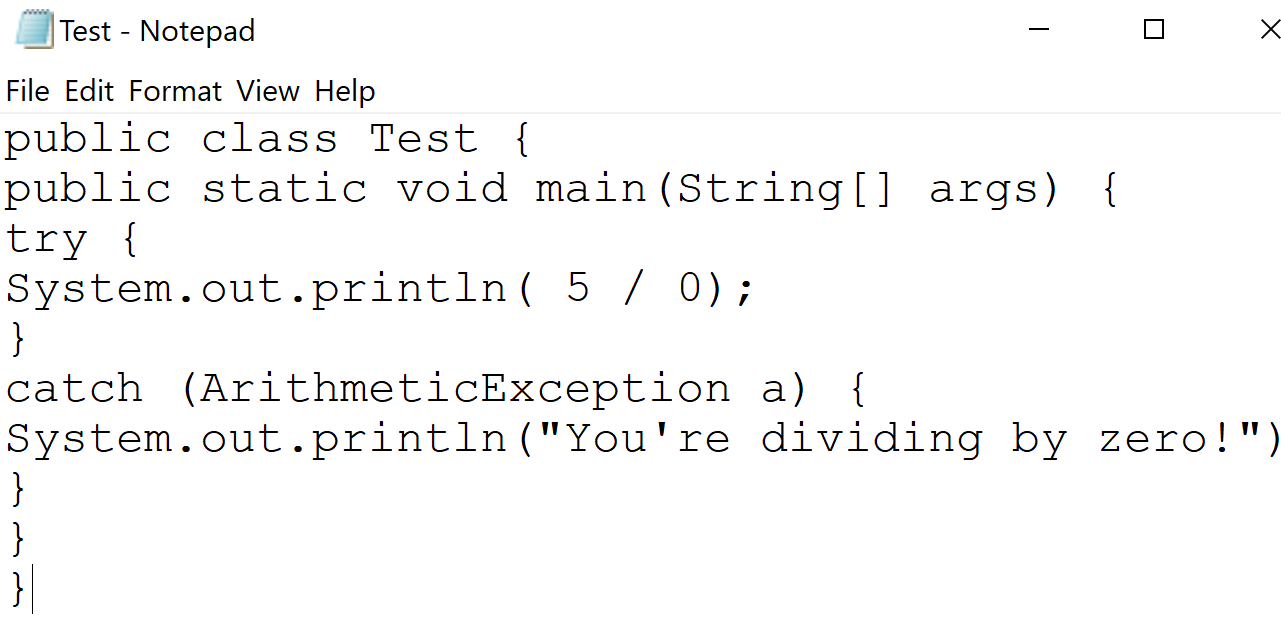
# 39) Example: A Little Demo.



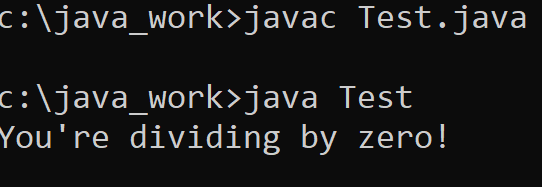
Result)



# 40) Test.java extension.



Result)



# 41) Using Objects To Represent Errors.

## • Java uses Exception objects to represent errors that happen during the exception execution of your program.

─ The moment of Exception type of an object creation (throwing of Exception) is the moment of the interrupt/exception.

─ Different thread paths of execution is used when exception event occur, the program does not terminate with the default JVM’s handler.

─ Exception events may happen or not happen, meaning that assigned handlers may run or may not run, i.e., exceptions add to the dynamic nature of precisely unpredictable thread evolution.

─ Greater flexibility to accommodate new or changed requirement.

# 42) Exception Delivery.

• Exceptions are delivered to the program outside of normal execution.

• Special exception handler blocks describe how exception events are to be handled (how to respond to the exception).

• Code for normal execution does not have to be interlaced with code for error handling

# 43) What To Do in Exception Handler.

## • The job of the exception handler is to put the application back into a stable state.

• And, possibly, to record the problem or notify the user.

# 44) General Type of Exception.

## • There are many kinds of exceptions that can all be represented by one class,

## Exception class.

─ Exception class object creation (throwing) does not help identify which of many possible exceptions exactly took place.

# 45) Example: Specific Types of Exceptions.

## • There are many kinds of exceptions.

─ In Java, Exception sort of a class to represent I/O exceptions is called java.io.IOException.

• Every time you read from the standard input stream, an exception of this type may be thrown.

# 46) When is a Throws Clause Needed?

• A method must have a throws clause when:

## - Method code uses throw exception object instruction.

# 47) When is Throws Clause not Needed?

• Two types of exceptions may be thrown by any method.

• Error objects: represent catastrophic VM problems, which may happen at any time.

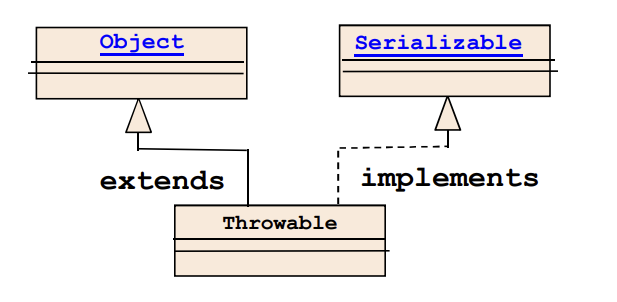
• Runtime Exceptions objects: Usually represent bugs.

• Since any method might have a bug in it, all methods are assumed by the JVM to possibly throw these.

# 48) Class Throwable.

• Only objects that are instances of Throwable class (or one of its subclasses) may be “thrown” by the Java Virtual Machine or can be thrown by the Java throw statement.

• Class Throwable or one of its subclasses can be the argument type in a catch clause.



# 49) Throwable Class.

• A Throwable contains a snapshot of the execution stack of its thread at the time it was created. • It can also contain a message string that gives more information about the error.

– Over time, a throwable can suppress other throwables from being propagated.

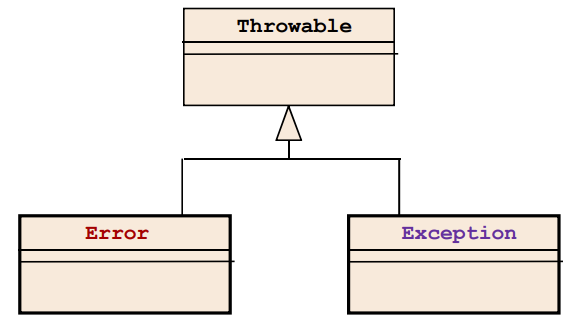
– Finally, the throwable can also contain a cause: another throwable that caused this throwable to be constructed.

– The recording of this causal information is referred to as the chained exception facility, as the cause can, itself, have a cause, and so on, leading to a "chain" of exceptions, each caused by another.

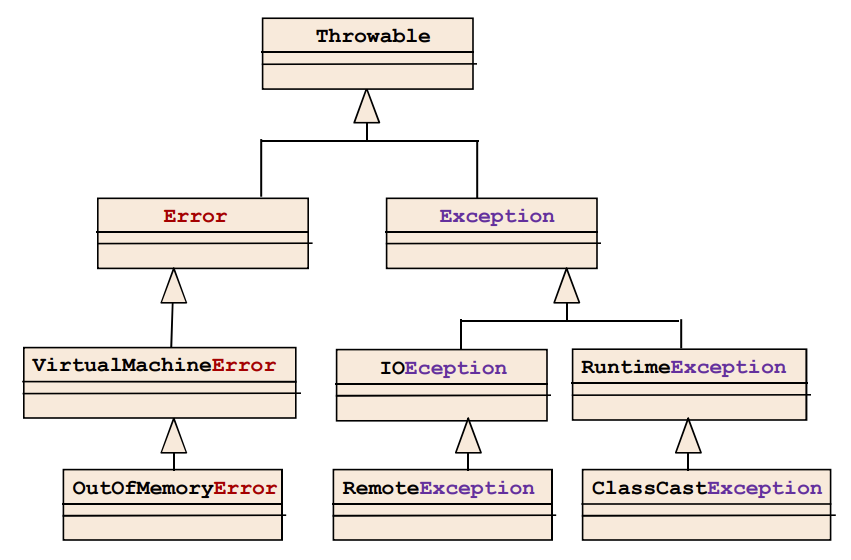
# 50) Java Exception Hierarchy.

## • Members of the Exception branch of the tree are thrown to indicate the abnormal conditions that can usually be handled by the application program.

• Members of the Error branch of the tree is reserved for more serious problems that occur at RT, (e.g. if no more memory is available, OutOfMemory error will occur, and there is nothing application can do about it.) JVM itself throws instances of Error or one of its subclasses.



## • Hierarchy tree of the Throwable class involves more than 100 direct or indirect subclasses and can be shown only partially.



# 51) Types (Sub Types) of Exceptions.

## • All exceptions in Java are objects of Throwable class.

• Unchecked Exceptions

– are exceptions derived from Error and RuntimeException classes.

– are usually irrecoverable and not handled explicitly.

– are not checked by the compiler.

• Checked Exceptions

– are exceptions derived from Exception class excluding the RuntimeException class.

– must be handled explicitly.

– are checked by the compiler.

# 52) Unchecked Exceptions.

## • Class Error & RunTimeException.

• Serious errors not handled by typical program.

• Usually indicate logic errors

– Example:

NullPointerException,

IndexOutOfBoundsException

• Catching unchecked exceptions is optional.

• Handled by Java Virtual Machine if not caught.

• Class Exception (except RunTimeException).

• Errors typical program should handle.

• Used for operations prone to error.

• Example

– IOException, ClassNotFoundException

• Compiler requires “catch or declare”.

– Catch and handle exception in method, OR

– Declare method can throw exception, force calling function to catch or declare exception in turn.

– Example

void A() throws ExceptionType{ … }

# 53) Handling Exceptions in Java.

## • Exception mechanism is built around the throw-and-catch paradigm.

– “to throw” means an exception has occurred • (Exception type of an object is created).

– “to catch” means to deal with an exception.

• If an exception is not caught, it is propagated to the call stack until a handler is found.

• Propagating an exception is called “ducking” the exception, or “passing the buck”.

# 54) System and Application Exceptions.

• Java exceptions can also be separated into two additional groups:

• system exceptions and

• application exception

# 55) System Exceptions.

• System exceptions are more serious in nature:

─ System exceptions are not related to the application logic and

─ Application is not expected to recover from such exceptions.

• System exceptions are mostly not checked:

─ As non-programming errors application is not supposed to catch them.

─ As severe errors about which nothing can be done. application cannot recover from and is not trying to catch.

# 56) Application Exceptions.

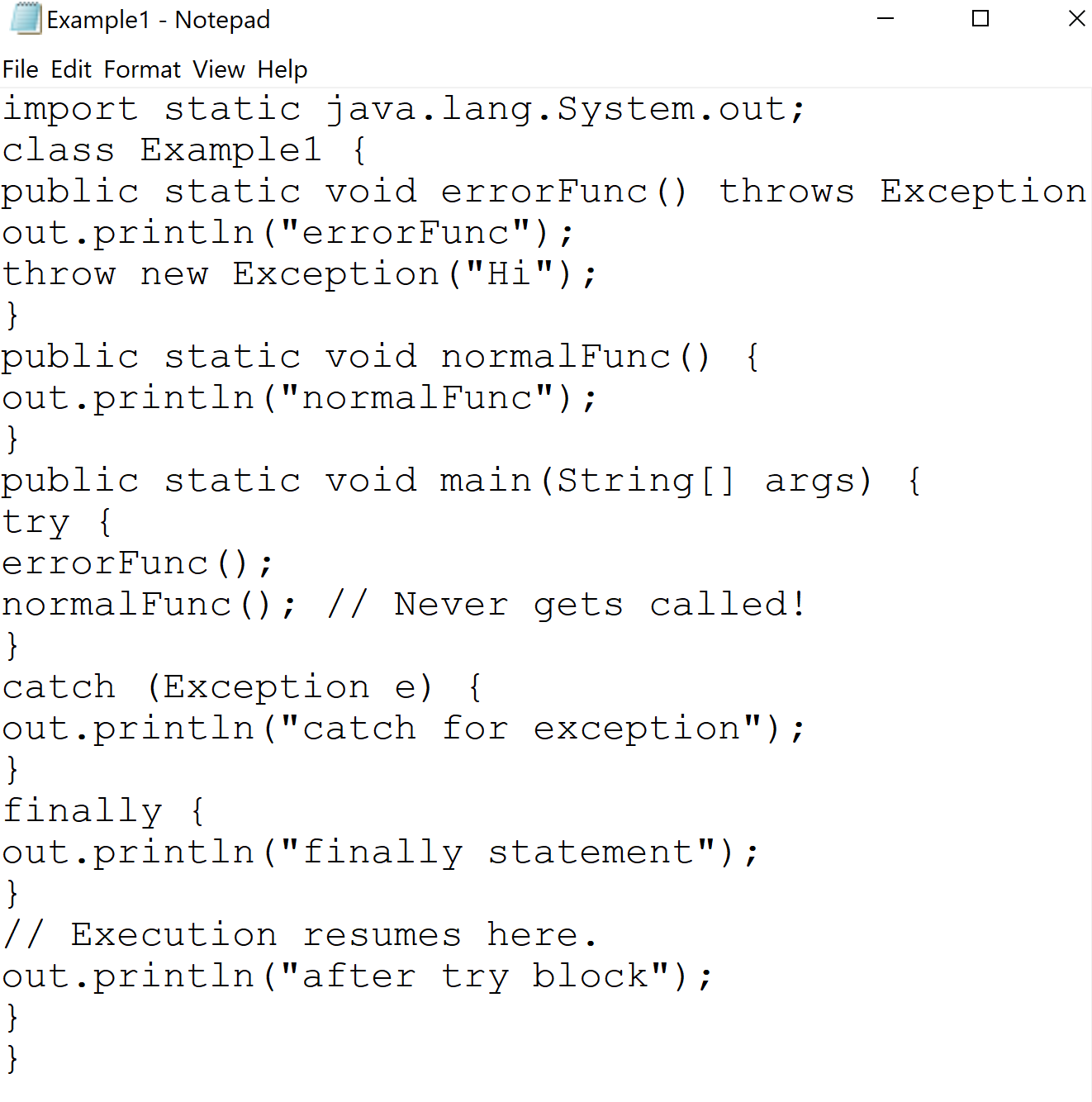
## • Application exceptions are not catastrophic in nature.

• Application exceptions are mostly checked and handled:

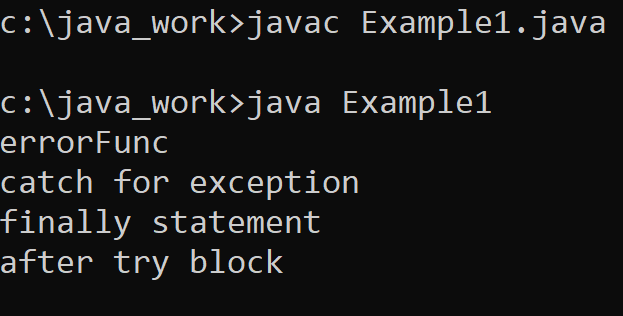
─ if recovery is possible the handler continues program execution (e.g user is asked to enters correct password), and

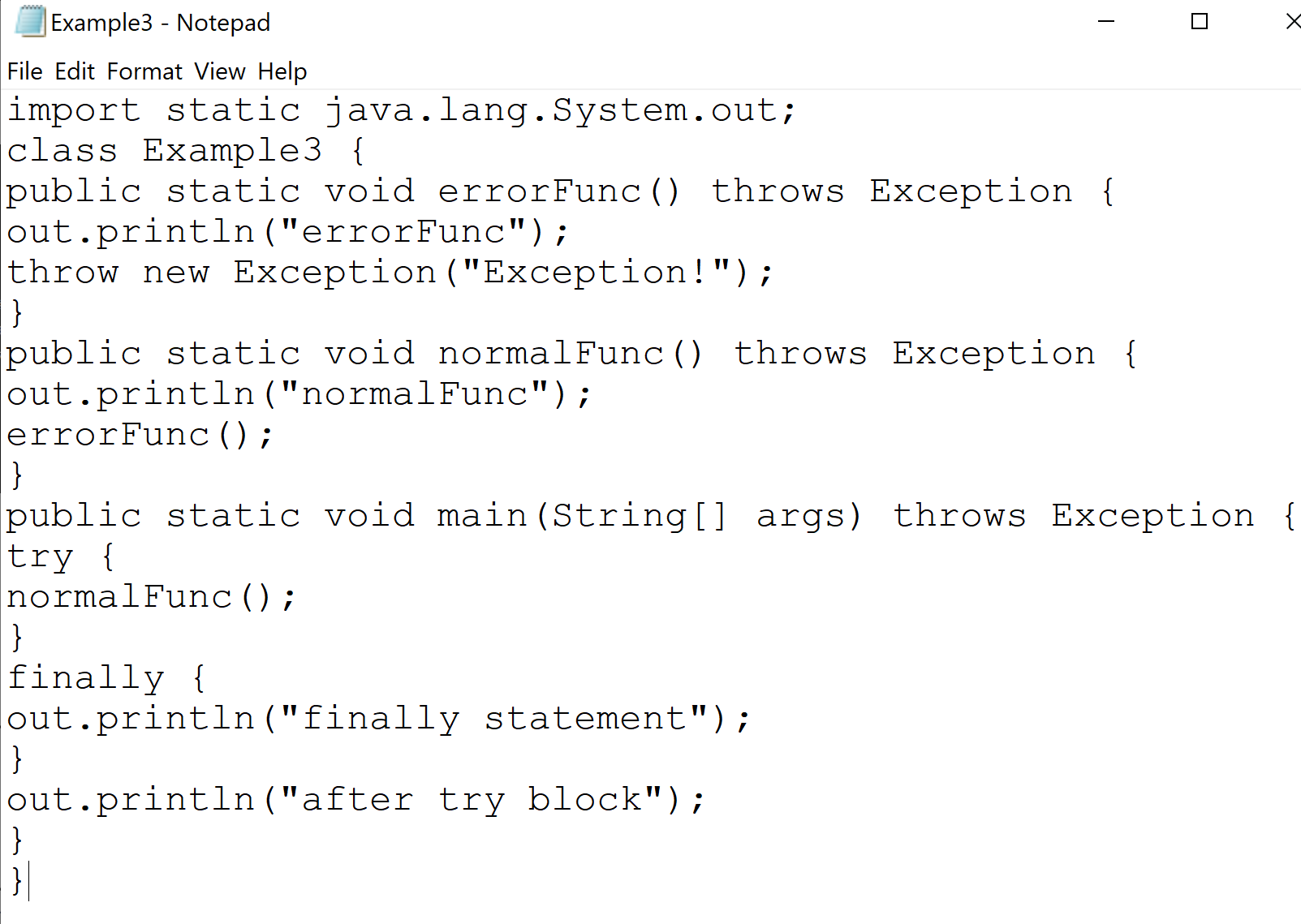
─ if recovery is not possible an error message is displayed and current application module is terminated.

# 57) Example: Uncaught Exception.

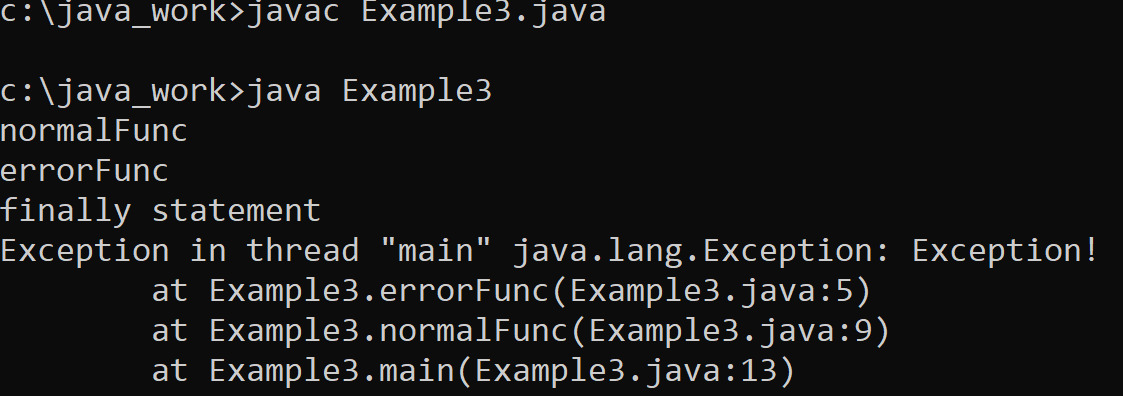


Result)

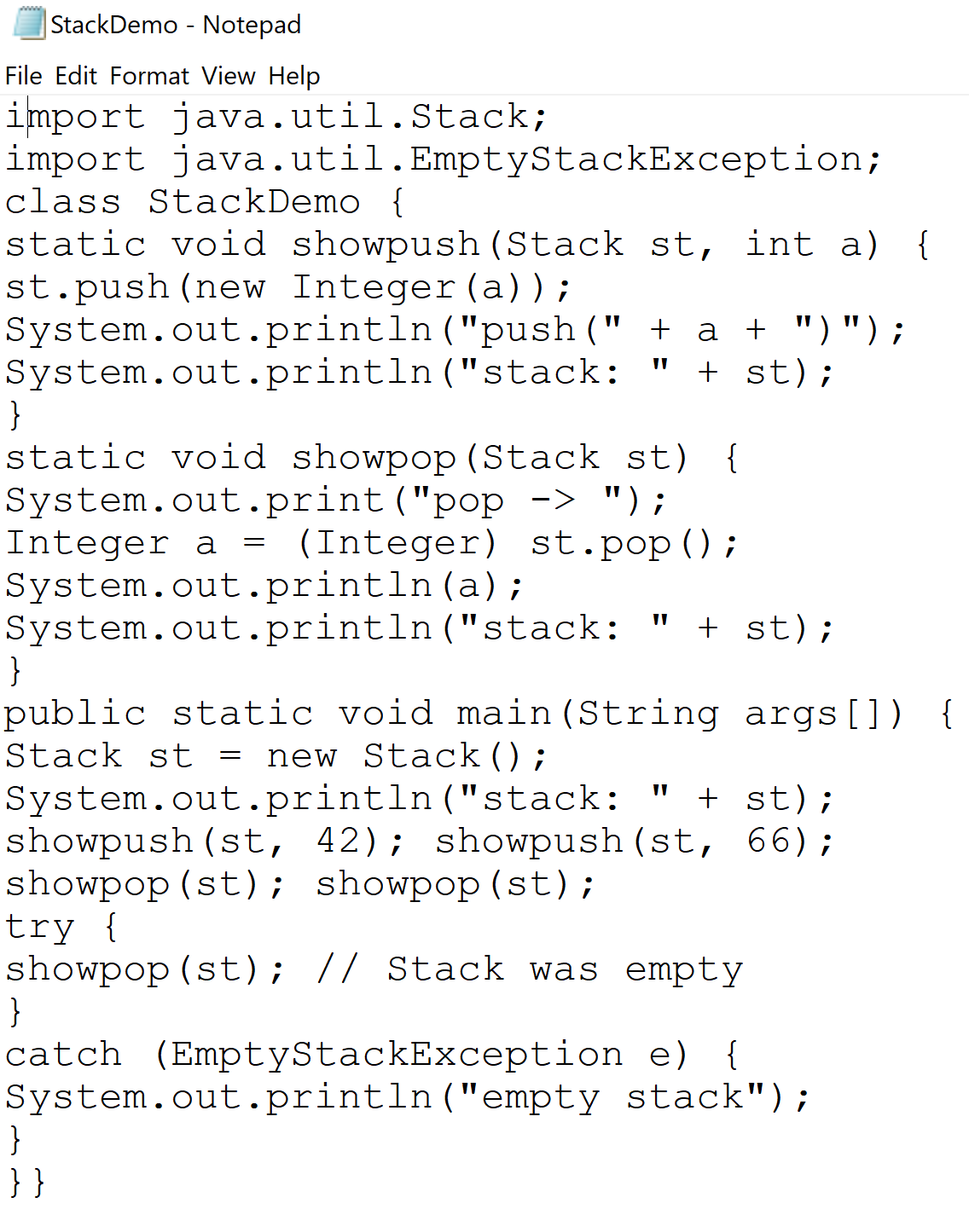




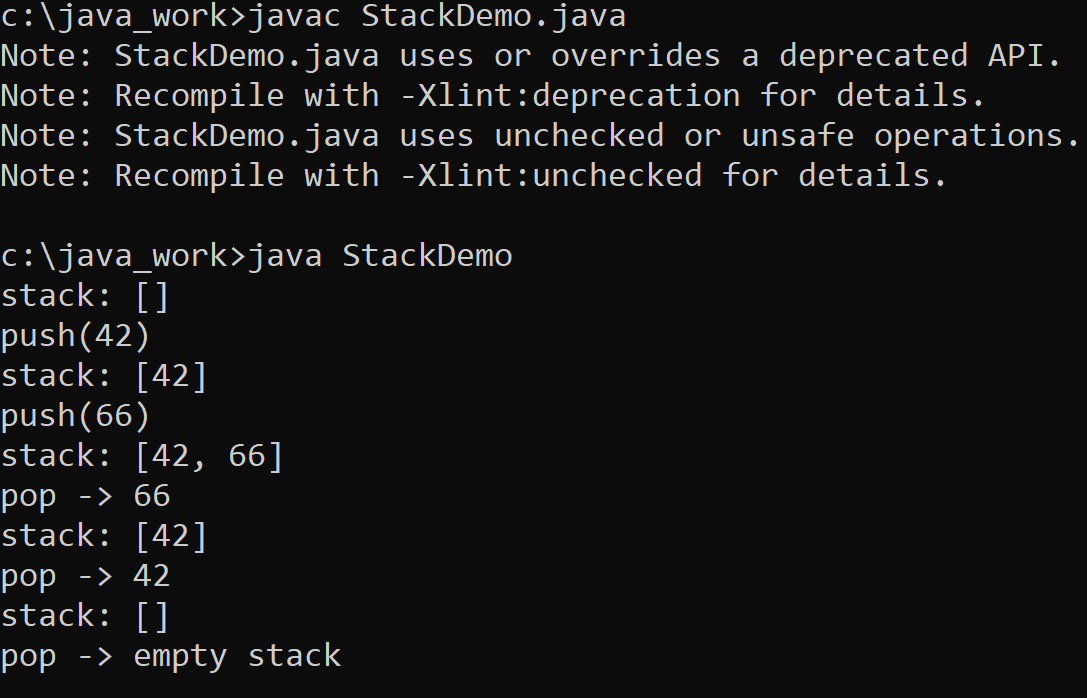
Result)



# 58) StackDemo.



Result)



# References

[1] http://tfbor.com/02\_725/05\_a\_SE\_AbstrctCls\_Interface/

[2] http://tfbor.com/02\_725/05\_b\_Exceptoins\_/

[3] R. (2021, July 26). *Java Exceptions Hierarchy Explained*. Rollbar. https://rollbar.com/blog/java-exceptions-hierarchy-explained/

[4] *Java - Exceptions*. (n.d.). TutorialsPoint. https://www.tutorialspoint.com/java/java\_exceptions.htm