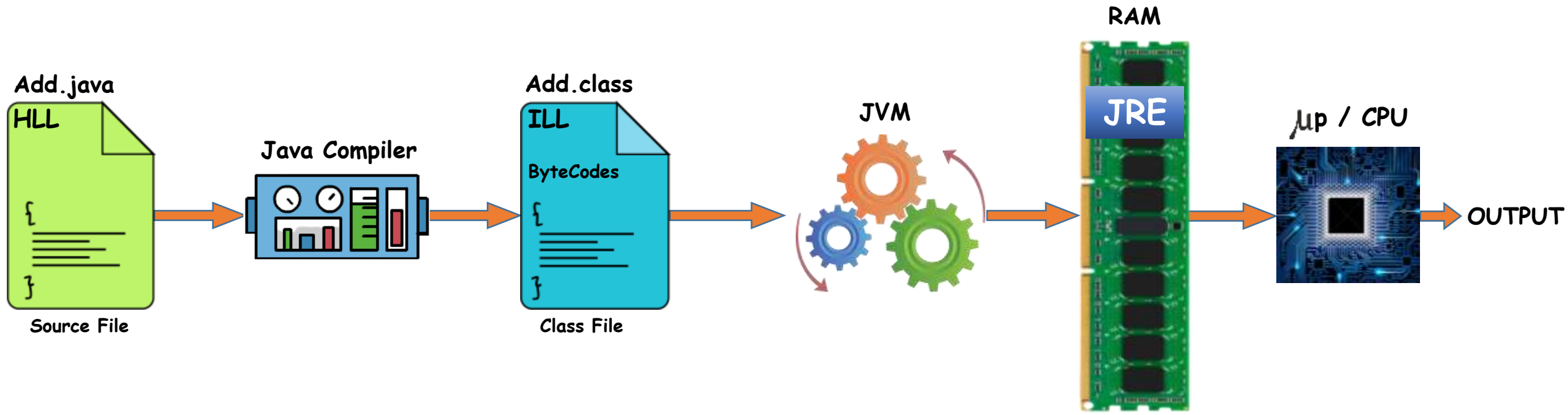
Exception Handling in Java

By Syed Zabi Ulla Sir,
Java & DSA Mentor,
PW Skills



Compilation Phase / Compilation Time

Compile Time Mistakes / Compile Time Errors

Faulty Coding By Programmer

Execution Phase / Execution Time / Runtime

Execution Time Mistakes / Exceptions

Faulty Inputs By User

Syntax Mistakes

Logical Mistakes

Q What is an exception?

Exceptions refer to logical mistakes that occur during the execution of a program.

Q What causes a program to generate an exception?

Faulty inputs provided by the users leading to logical mistakes in the program.

Q When does an exception occur?

During the execution of the program.

Q What happens if an exception occurs?

It disturbs the normal flow of the program resulting in abrupt termination.
Hence, valuable resources may be lost.

Q What is meant by exception handling?

Exception handling refers to the process of handling the exception object in such a manner that it prevents abrupt termination. The main objective of exception handling is graceful termination of the program.

Q List the keywords related to exception handling.

try - catch - finally - throw - throws

Types Of Architecture

1940s

1950s

1960s

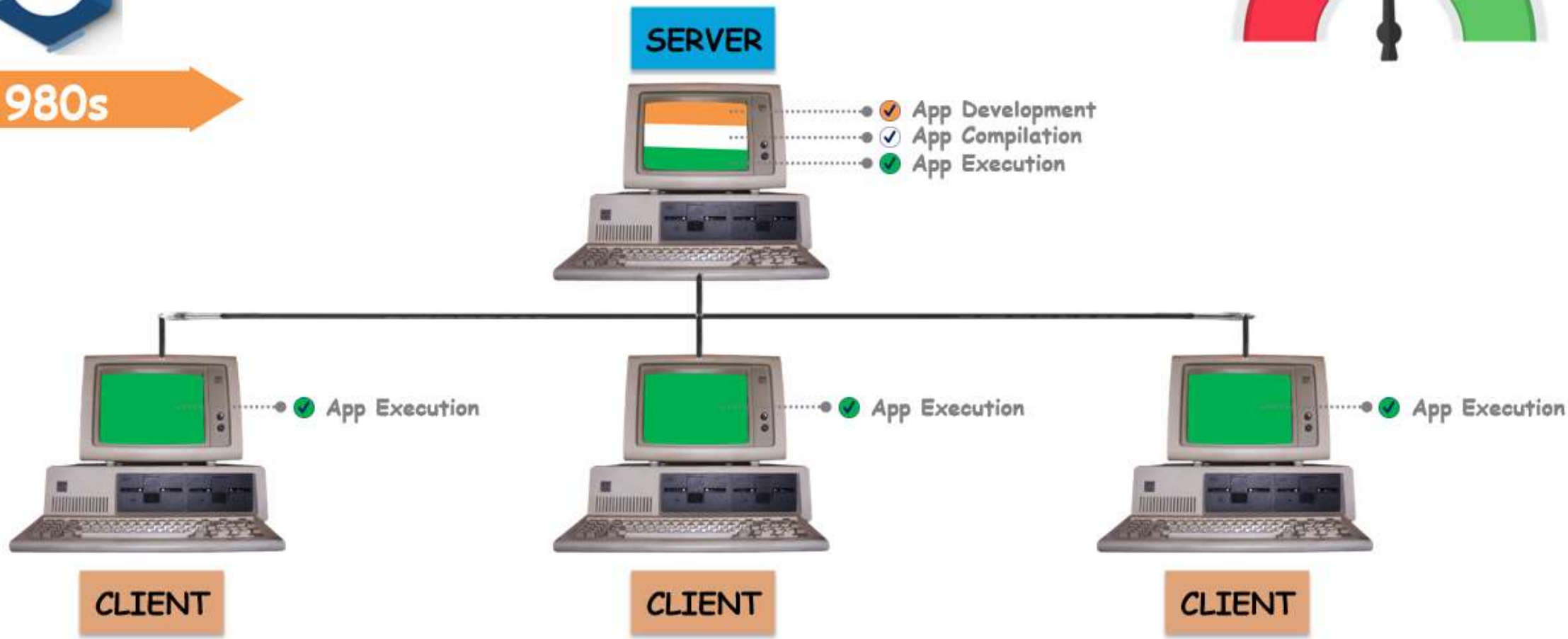
1970s

- App Development
- App Compilation
- App Execution

1-Tier Architecture

Stand-Alone Architecture

1980s



2-Tier Architecture

Client-Server Architecture

Java

1990s

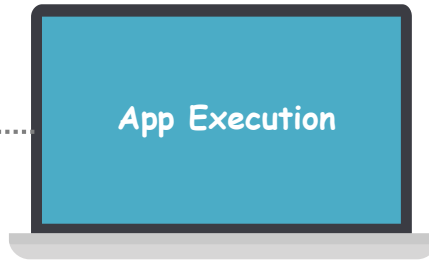
App Development
App Compilation
App Execution

Upload

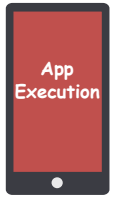
Download

Download

Download



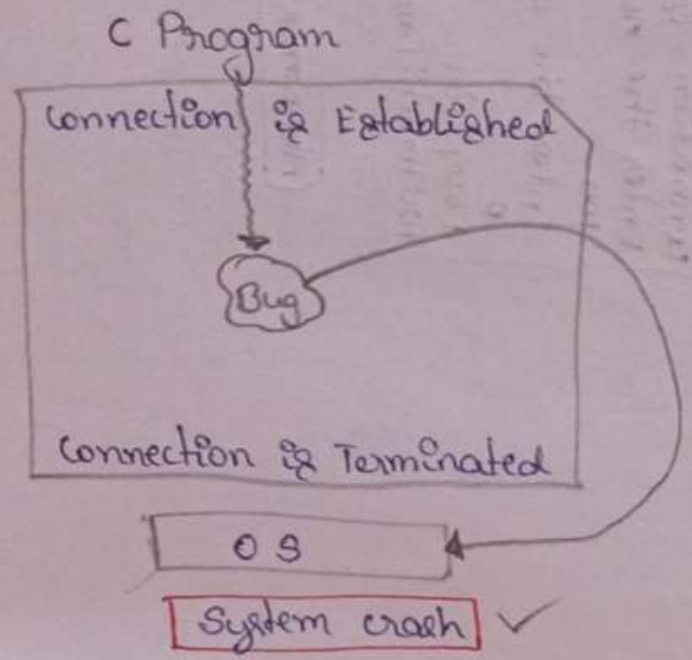
Download

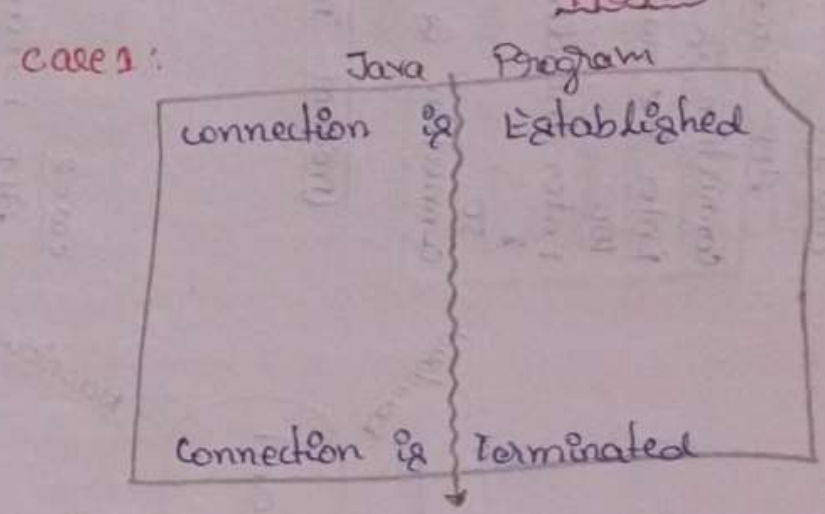


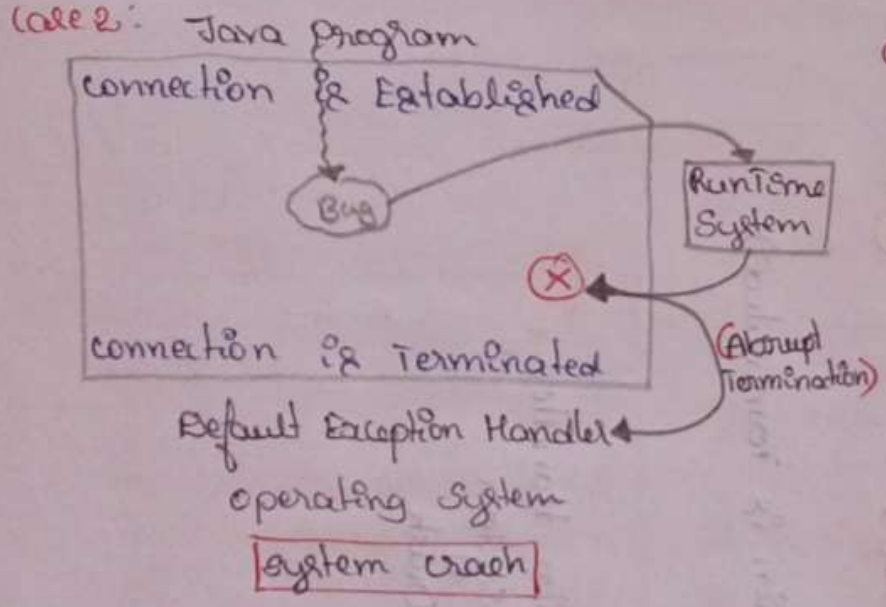
Download

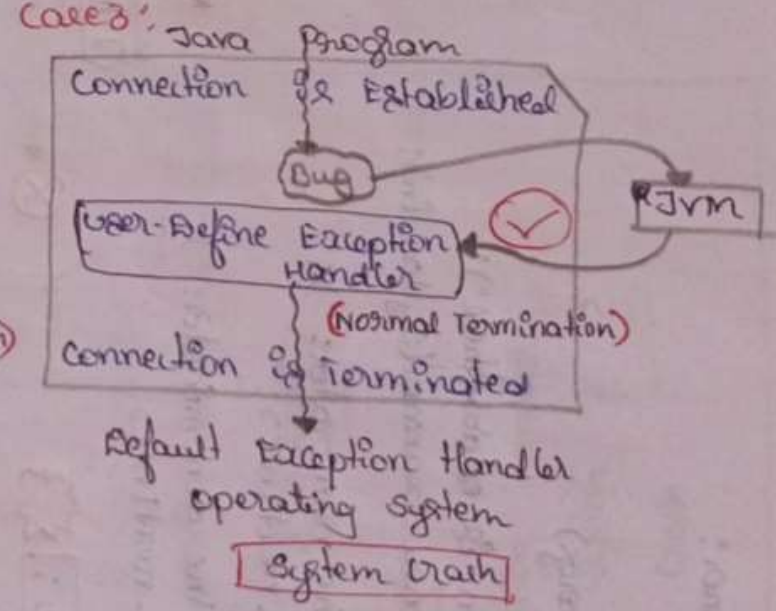
N-Tier Architecture

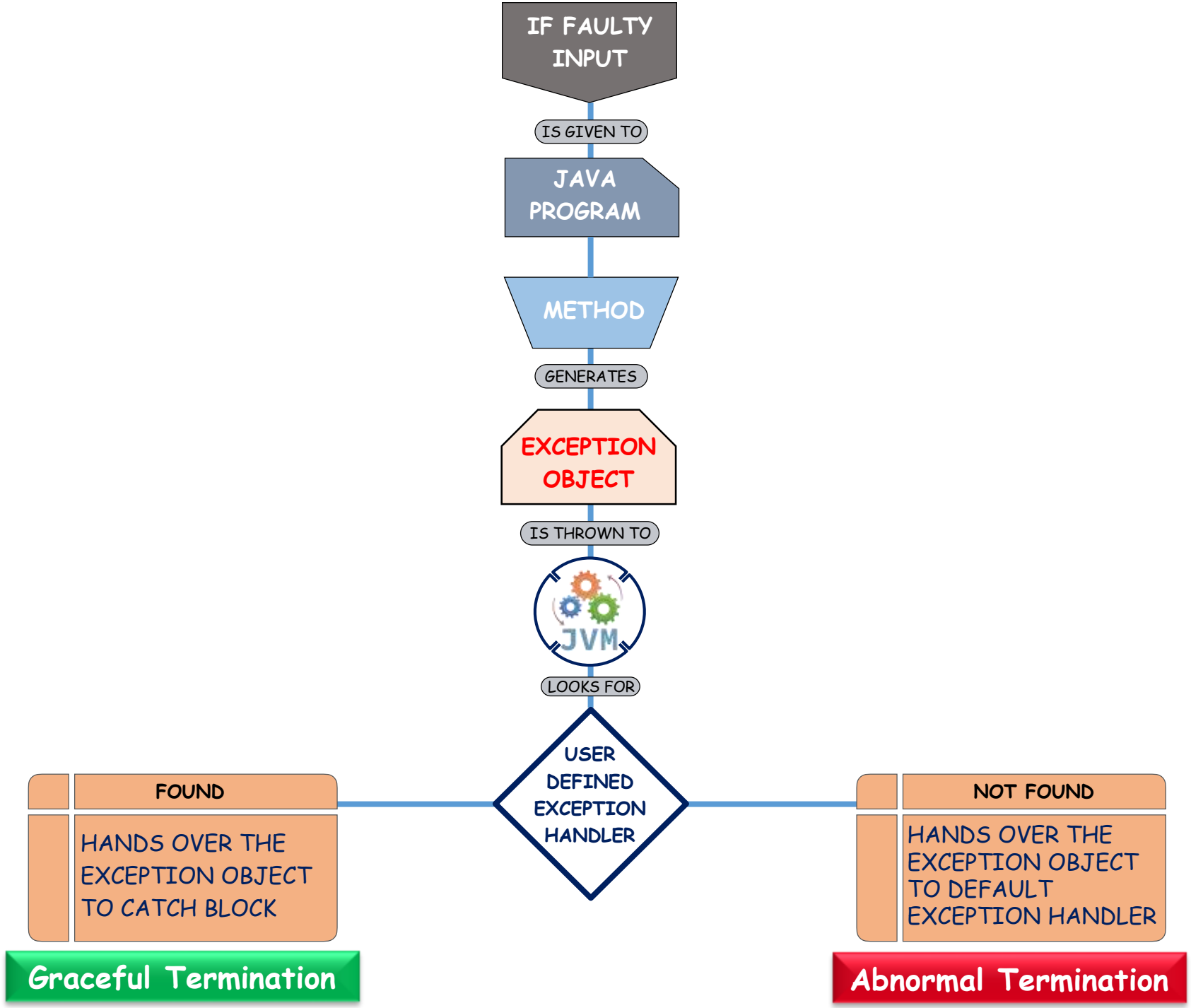
Internet Architecture


C Program
connection is Established
Bug
connection is Terminated
OS
System crash


Case 1:
Java Program
connection is Established
connection is Terminated


Case 2:
Java program
connection is Established
Bug
RunTime System
connection is Terminated
(Abrupt Termination)
Default Exception Handler
operating system
system crash


Case 3:
Java Program
Connection is Established
Bug
JVM
User-Define Exception Handler
(Normal Termination)
Connection is Terminated
Default Exception Handler
operating system
System crash


IF FAULTY INPUT
IS GIVEN TO
JAVA PROGRAM
METHOD
GENERATES
EXCEPTION OBJECT
IS THROWN TO
JVM
LOOKS FOR
USER DEFINED EXCEPTION HANDLER
FOUND
HANDS OVER THE EXCEPTION OBJECT TO CATCH BLOCK
Graceful Termination
NOT FOUND
HANDS OVER THE EXCEPTION OBJECT TO DEFAULT EXCEPTION HANDLER
Abnormal Termination

Different ways of handling the Exception:

1. Handling the Exception [try-catch]
2. Rethrowing the Exception [try-catch-throw-throws-finally]
3. Ducking the Exception [throws]

Different ways of handling the exception:

1) Handling the Exception [try-catch]
2) Rethrowing the Exception [try-catch-throw-throws-finally]
3) Backing the Exception [throws]

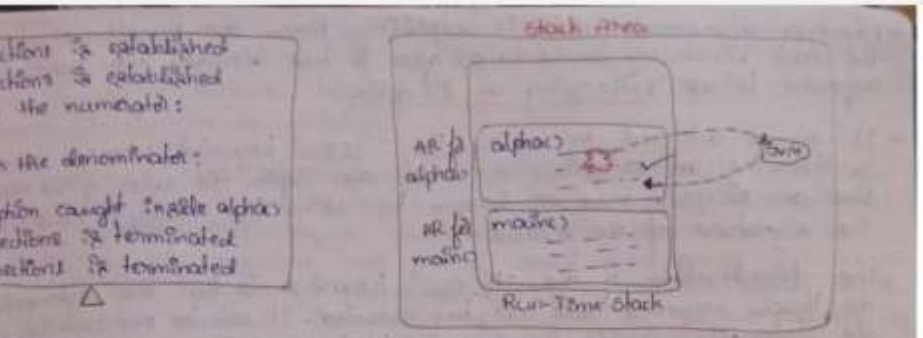
1) Handling the exception:

```
class Demo
{
  public void alpha()
  {
    S.o.p("connection is established");
    Scanner scan = new Scanner(System.in);
    try
    {
      S.o.p("Enter the numerator:");
      int a = scan.nextInt();
      S.o.p("Enter the denominator:");
      int b = scan.nextInt();
      int c = a/b;
      S.o.p(c);
    }
    catch(Exception e)
    {
      S.o.p("Exception caught inside alpha");
    }
    S.o.p("connections is terminated");
  }
}

class Launch
{
  psvm(String[] args)
  {
    S.o.p("connections is established");
    Demo d = new Demo();
    d.alpha();
    S.o.p("connections is terminated");
  }
}
```

JVM

```
connections is established
connections is established
Enter the numerator:
100
Enter the denominator:
0
Exception caught inside alpha
connections is terminated
connections is terminated
```

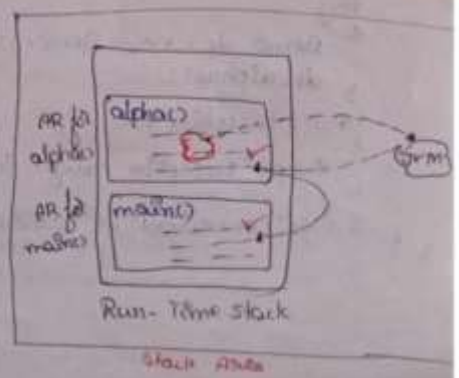


- Handling the exception object within the same method inside which exception occur is only refered to as "handling the exception".
- As noticed in the above program, if an exception is handled inside a method then JVM will not automatically propagate the exception object down the stack hierarchy.

2) Rethrowing the exception:

```
class Demo
{
  public void alpha() throws Exception
  {
    S.o.p("connections is established");
    Scanner scan = new Scanner(System.in);
    try
    {
      S.o.p("Enter the numerator:");
      int a = scan.nextInt();
      S.o.p("Enter the denominator:");
      int b = scan.nextInt();
      int c = a/b;
    }
    S.o.p(c);
    catch(Exception e)
    {
      S.o.p("Exception caught inside alpha()");
      throw e;
    }
    finally
    {
      S.o.p("connections is terminated");
    }
  }
}
```

```
class Launch                    JVM
{
  psvm(String[] args)
  {
    S.o.p("connections is established");
    try
    {
      Demo d1 = new Demo();
      d1.alpha();
    }
    catch(Exception e)
    {
      S.o.p("Exception caught inside
             main()");
    }
    S.o.p("connections is terminated");
  }
}
```




```
o/p
connections is established
connections is established
Enter the numerator:
10
Enter the denominator:
0
Exception caught inside alpha()
connections is terminated
Exception caught inside main()
connections is terminated
```

- Whenever a method wants to explicitly throw the exception object down the stack hierarchy (to the caller) after it has already been handled is referred to as "Rethrowing an exception".
- It can be achieved by using the "throw" keyword. Whenever a method is throwing an exception, it must also inform the caller of the method that an exception may be thrown by using the "throws" keyword in the signature of the method.
- The disadvantage of the "throw" keyword is that the statements below the throw keyword will not be executed. It can be overcome by placing the statements inside the "finally" block.

3) Ducking the Exception:

```
class Demo
{
    public void alpha() throws Exception
    {
        S.o.p("connections is established");
        Scanner scan = new Scanner(System.in);
        try
        {
            S.o.p("Enter the numerator: ");
            int a = scan.nextInt();
            S.o.p("Enter the denominator: ");
            int b = scan.nextInt();
            int c = a/b;
            S.o.p(c);
        }
        finally
        {
            S.o.p("connections is terminated");} → Critical statement must be executed
    }
}
class Launch
{
    p s v m(String[] args)
    {
        S.o.p("connection is established");
        try
        {
            Demo d1 = new Demo();
            d1.alpha();
        }
        catch (Exception e)
        {
            S.o.p("Exception caught Inside main");
        }
        S.o.p("connections is terminated");
    }
}
```

```
connection is established
connections is established
Enter the numerator:
100
Enter the denominator:
0
connections is terminated
Exception caught Inside main
connections is terminated
```

Whenever a method does not want to handle any exception and instead handover the responsibility of handling the exception to the caller of the method by using throws keyword in the signature of the method, this is known as ducking an exception

As noticed in the above program, within the alpha() method catch block had not been provided. Since the alpha() method does not want to handle the exception. Incase exception occurs, alpha() method be would be abruptly executed and control would come out of alpha() method but if there are any critical statements within the alpha() method that should be executed irrespective of whether or not an exception occurred. Then such statements must be placed under the 'finally' block. However, it is invalid to have only a 'finally' block without the 'try' block. Hence 'finally' must always be used atleast with the 'try' block