# Learning Java - A Foundational Journey

Session: 9

Exceptions

## Objectives

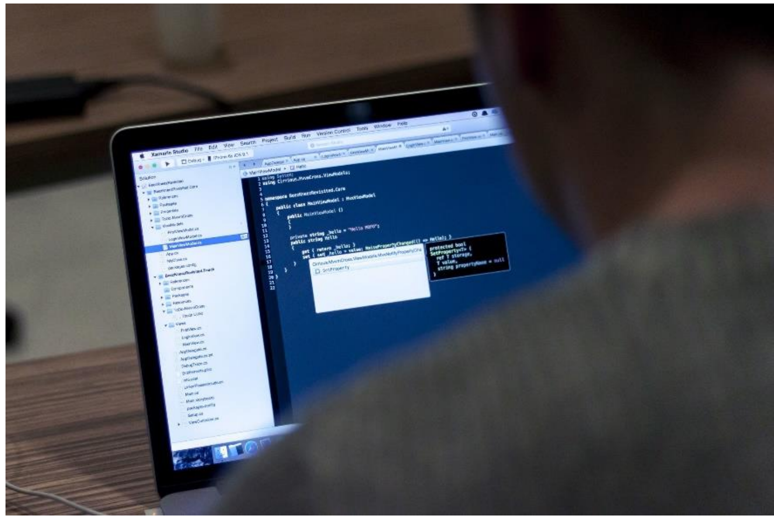
- Describe exceptions
- Explain types of errors and exceptions
- Describe the Exception class
- Describe exception handling
- Explain try-catch block
- Explain finally block
- Explain execution flow of exceptions
- Explain guidelines for exception handling

## Introduction

- Java is a robust and efficient programming language.
- Features such as classes, objects, inheritance, and so on make Java a strong, versatile, and secure language.
- However, no matter how well a code is written, it is prone to failure or behaves erroneously in certain conditions.
- Java provides the concept of exception handling using which a programmer can display appropriate message to the user in case such unexpected behavior of code occurs.

## Exceptions 1-3

- Exception:
  - Is an event or an abnormal condition in a program occurring during execution of a program leading to disruption of normal flow of program instructions.
  - Can occur for different reasons such as:
    - User enters invalid data
    - A file that must be opened cannot be found
    - A network connection has been lost in the middle of communications
    - JVM has run out of memory

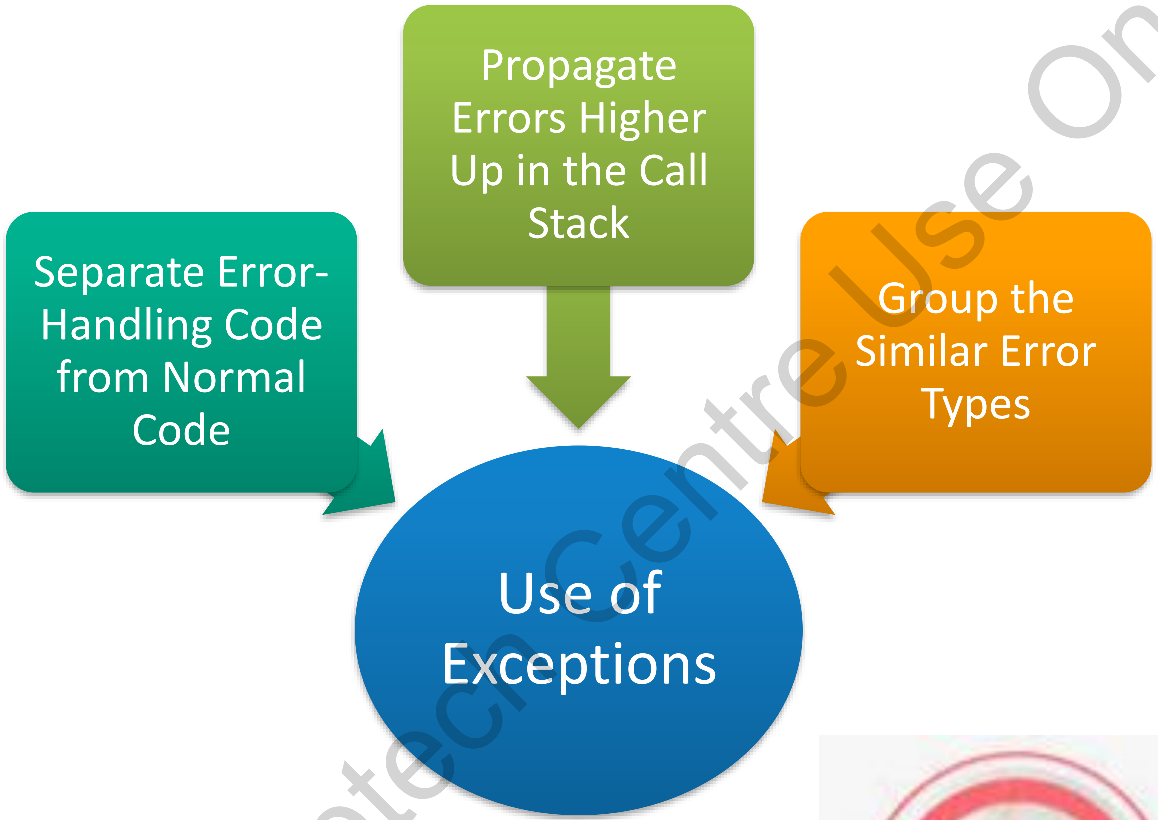


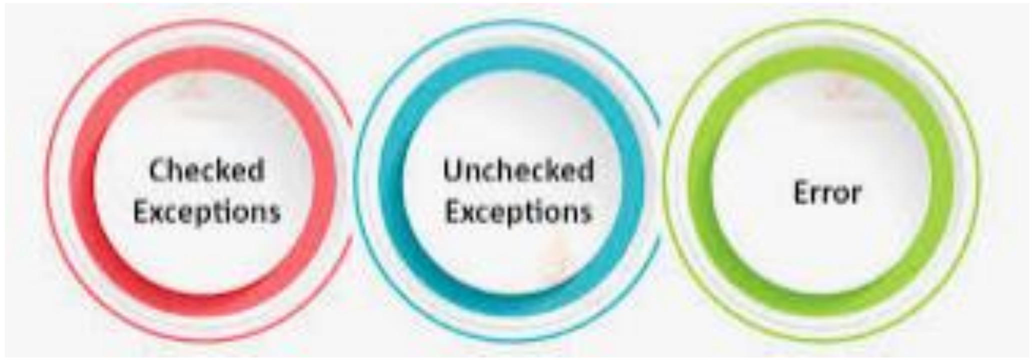
## Exceptions 2-3

An exception is thrown for following reasons:

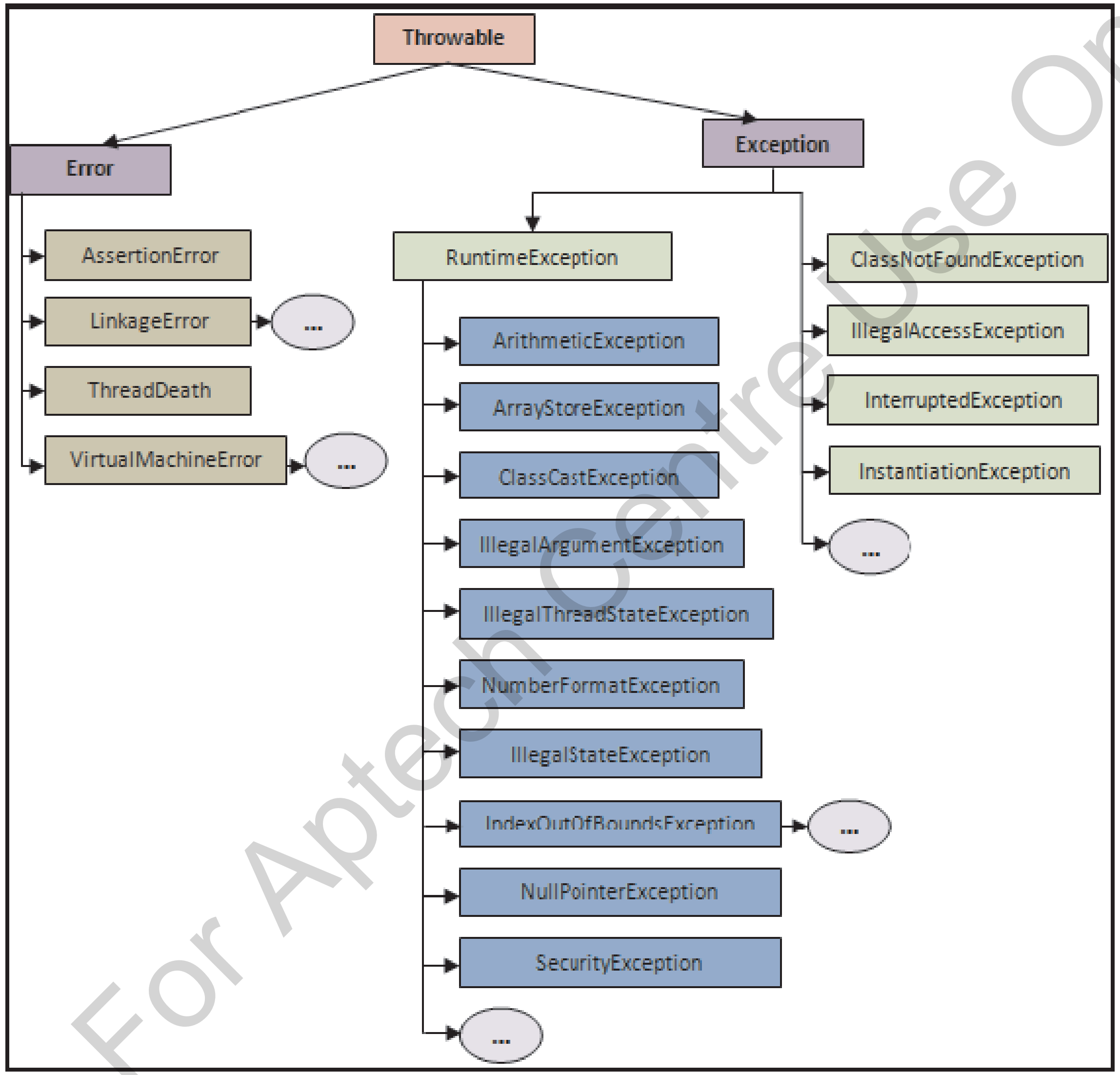
- A `throw` statement within a method was executed
- An abnormality in execution was detected by the JVM
  - Violation of normal semantics of Java while evaluating an expression
  - Error occurring while linking, loading, or initializing part of the program
  - JVM is prevented from executing the code due to an internal error or resource limitation
- An asynchronous exception occurred

## Exceptions 3-3


Propagate Errors Higher Up in the Call Stack
Separate Error-Handling Code from Normal Code
Group the Similar Error Types
Use of Exceptions



# Types of Errors and Exceptions 1-2



# Types of Errors and Exceptions 2-2

| Exception | Description |
|---|---|
| `InstantiationException` | Occurs upon an attempt to create instance of an `abstract` class. |
| `InterruptedException` | Occurs when a thread is interrupted. |
| `NoSuchMethodException` | Occurs when JVM is unable to resolve which method is to be invoked. |
| `AArithmeticException` | Indicates an arithmetic error condition. |
| `ArrayIndexOutOfBoundsException` | Occurs if an array index is less than zero or greater than the actual size of the array. |
| `IllegalArgumentException` | Occurs if method receives an illegal argument. |
| `NegativeArraySizeException` | Occurs if array size is less than zero. |
| `NullPointerException` | Occurs on access to a `null` object member. |
| `NumberFormatException` | Occurs if unable to convert the string to a number. |
| `StringIndexOutOfBoundsException` | Occurs if index is negative or greater than the size of the string. |

# Exception Class 1-2

The `Exception` class and all its subclasses except `RuntimeException`

The checked exceptions must be declared in a method or constructor's `throws` clause

**Syntax**

```
public class Exception extends Throwable
{
 …
}
```

| Exception Class Constructor | Description |
|---|---|
| `Exception()` | Constructs a new exception with error message set to `null`. |
| `Exception(String message)` | Constructs a new exception with error message set to the specified string `message`. |
| `Exception(String message, Throwable cause)` | Constructs a new exception with error message set to the specified strings `message` and `cause`. |
| `Exception(Throwable cause)` | Constructs a new exception with the specified `cause`. The error message is set as per the evaluation of `cause == null?null:cause.toString()`. That is, if `cause is null`, it will return `null`, else it will return the `String` representation of the message. The message is usually the class name and detail message of `cause`. |

# Exception Class 2-2

| Exception Class Method | Description |
|---|---|
| `public String getMessage()` | Returns the details about the exception that has occurred. |
| `public Throwable getCause()` | Returns the cause of the exception that is represented by a `Throwable` object. |
| `public String toString()` | If the `Throwable` object is created with a message string that is not `null`, it returns the result of `getMessage()` along with the name of the exception class concatenated to it. If the `Throwable` object is created with a `null` message string, it returns the name of the actual class of the object. |
| `public void printStackTrace()` | Prints the result of the method, `toString()` and the stack trace to `System.err`, that is, the error output stream. |
| `public StackTraceElement [] getStackTrace()` | Returns an array where each element contains a frame of the stack trace. The index 0 represents the method at the top of the call stack and the last element represents the method at the bottom of the call stack. |
| `public Throwable fillInStackTrace()` | Fills the stack trace of this `Throwable` object with the current stack trace, adding to any previous information in the stack trace. |

## Handling Exceptions in Java

Any exception that a method is liable to throw is considered to be as much a part of that method's programming interface as its parameters and return value

More than one runtime exceptions can occur anywhere in a program

A common situation where a user can throw a `RuntimeException` is when the user calls a method incorrectly

For example, a method can check beforehand if one of its arguments is incorrectly specified as `null`. In that case, the method may throw a `NullPointerException`, which is an unchecked exception

## try-catch Block 1-2

- The first step in creating an exception handler is to identify code that may throw an exception and enclose it within `try` block.

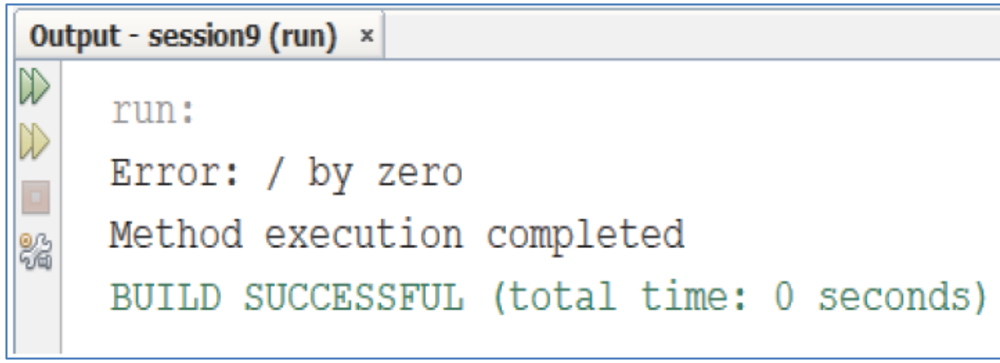
**Syntax:**

```
try{
          // statement 1
          // statement 2
}
```

```
package session9;
class Mathematics {
/**
* Divides two integers
* @param num1 an integer variable storing value of first number
* @param num2 an integer variable storing value of second number
* @return void
*/
public void divide(int num1, int num2) {
// Create the try block
try {
// Statement that can cause exception
System.out.println("Division is: " + (num1/num2));
}
catch(ArithmeticException e){ //catch block for ArithmeticException
// Display an error message to the user
System.out.println("Error: "+ e.getMessage());
```

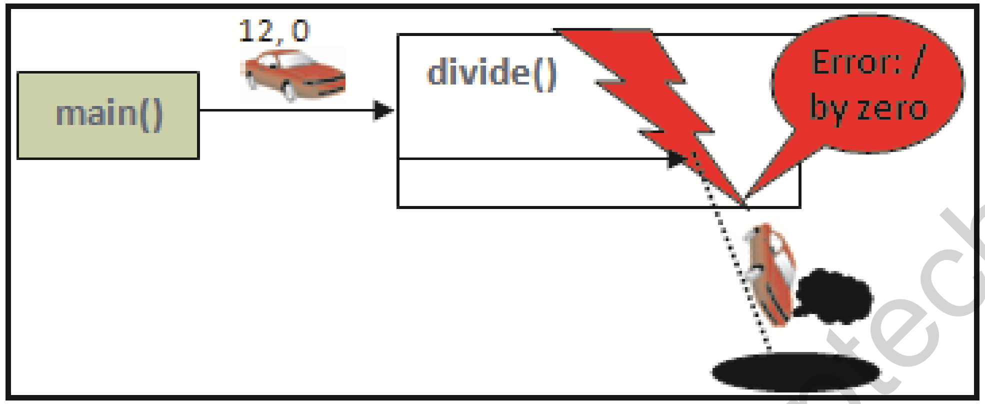
## try-catch Block 2-2

```
}
// Rest of the method
System.out.println("Method execution completed");
}
}
/**
* Define the TestMath.java class
*/
public class TestMath {
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
// Check the number of command line arguments
if(args.length==2) {
// Instantiate the Mathematics class
Mathematics objMath = new Mathematics();
// Invoke the divide(int,int) method
objMath.divide(Integer.parseInt(args[0]), Integer.parseInt(args[1]));
}
else {
System.out.println("Usage: java TestMath <number1> <number2>");
}
}
}
```

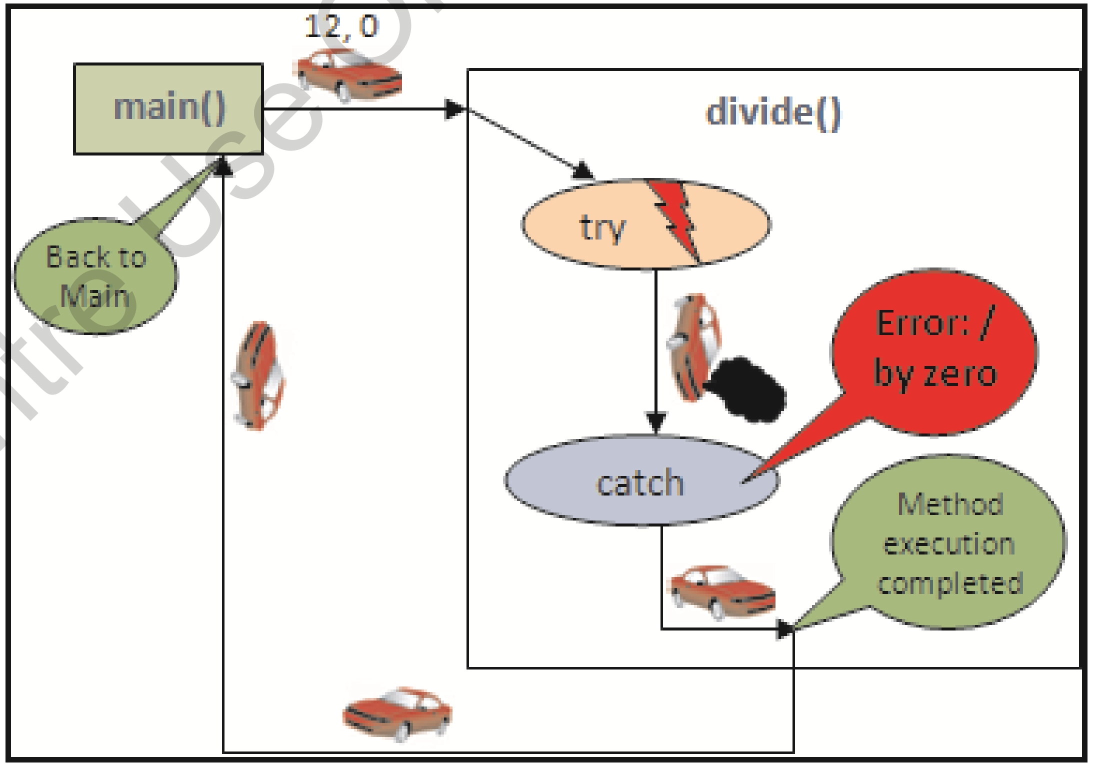

Output - session9 (run) ×

```
run:
Error: / by zero
Method execution completed
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Execution Flow of Exceptions



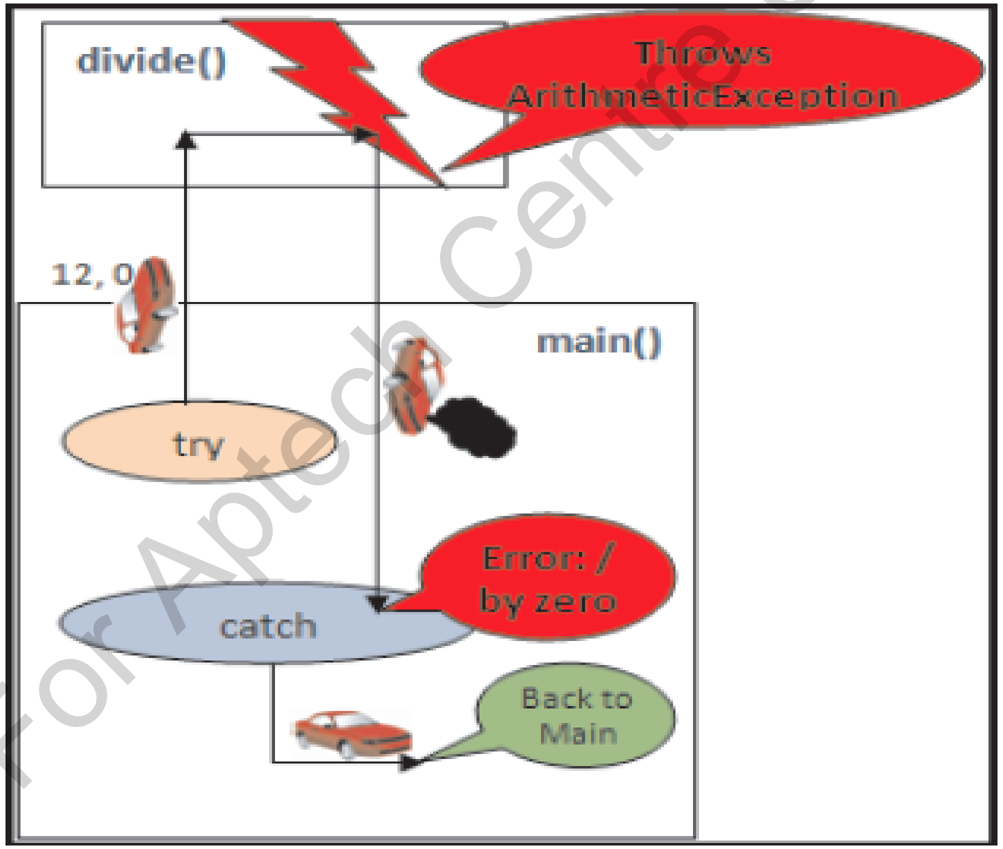
Execution Without try-catch Block



Execution of try-catch Block

## throw and throws Keywords

- Java provides `throw` and `throws` keywords to explicitly raise an exception in `main()` method.
- The `throw` keyword throws exception in a method.
- The `throws` keyword indicates exception that a method may throw.

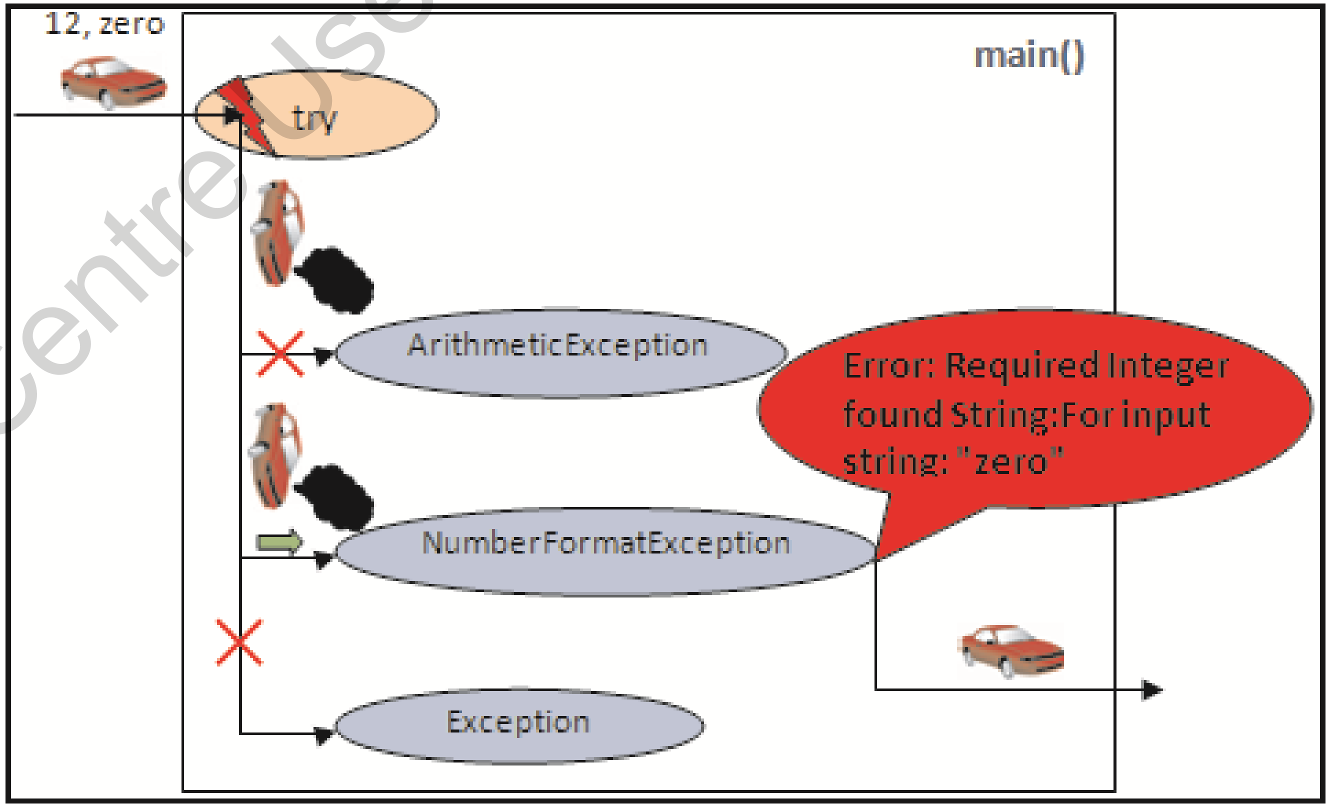


## Multiple catch Blocks

- The user can associate multiple exception handlers with a `try` block by providing more than one `catch` blocks directly after the `try` block.

**Syntax:**

```
try
{…}
catch (<exception-type>
<object-name>)
{…}
catch (<exception-type>
<object-name>)
{…}
```



## finally Block 1-2

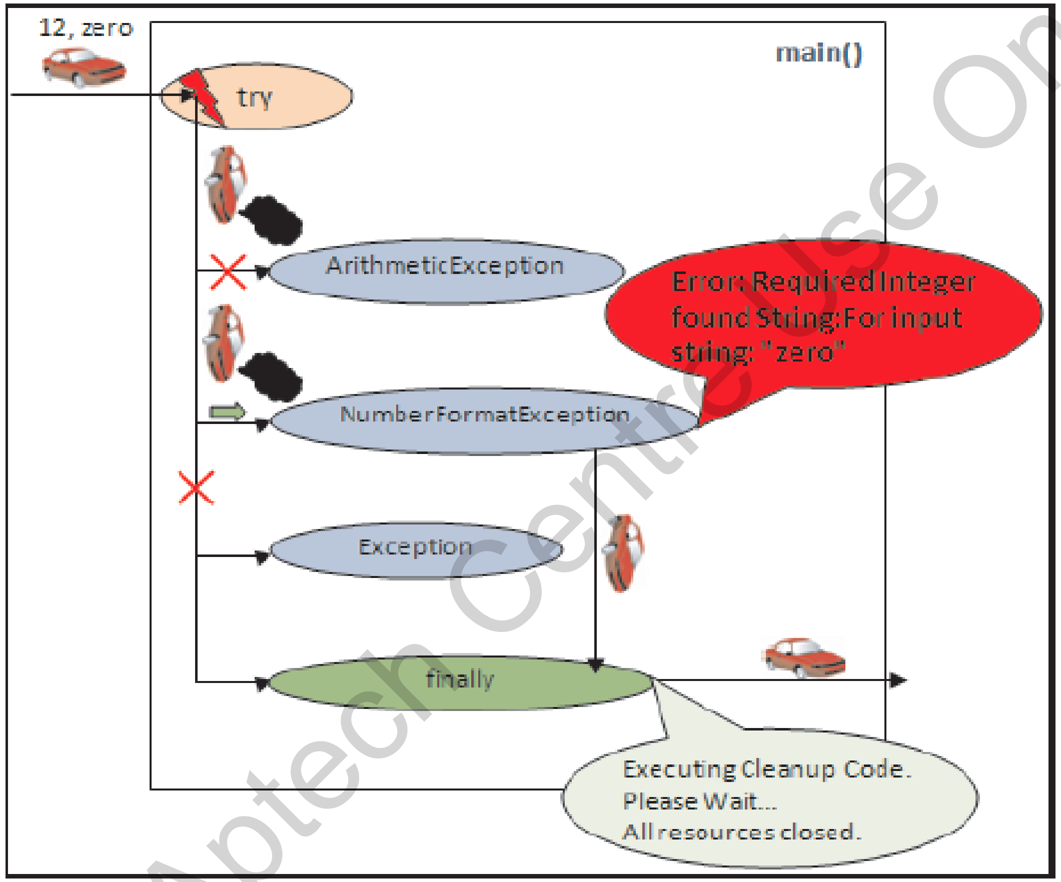
Java provides the `finally` block to ensure execution of certain statements even when an exception occurs.

Cleanup code is not accidentally bypassed by a `return`, `break`, or `continue` statement.

The `finally` block is mainly used as a tool to prevent resource leaks.

If the JVM exits while executing the `try` or `catch` block, then the `finally` block may not execute.

# finally Block 2-2


12, zero
main()
try
ArithmeticException
Error: Required Integer found String: For input string: "zero"
NumberFormatException
Exception
finally
Executing Cleanup Code. Please Wait... All resources closed.

## Guidelines for Handling Exceptions

The `try` statement must be followed by at least one `catch` or a `finally` block.

Use the `throw` statement to throw an exception that a method does not handle by itself.

The `finally` block must be used to write clean up code.

Provide appropriate message along with the default message when an exception occurs.

Exceptions should not be used to indicate normal branching conditions

## Summary

- An exception is an event or an abnormal condition in a program occurring during execution of a program that leads to disruption of the normal flow of the program instructions.
- The process of creating an exception object and passing it to the runtime system is termed as throwing an exception.
- An appropriate exception handler is one that handles the same type of exception as the one thrown by the method.
- Checked exceptions are exceptions that a well-written application must anticipate and provide methods to recover from.
- Errors are exceptions that are external to the application and the application usually cannot anticipate or recover from errors.
- Runtime Exceptions are exceptions that are internal to the application from which the application usually cannot anticipate or recover from.
- Throwable class is the base class of all exception classes and has two direct subclasses namely, Exception and Error.
- The try block is a block of code which might raise an exception and catch block is a block of code used to handle a particular type of exception.
- The user can associate multiple exception handlers with a try block by providing more than one catch blocks directly after the try block.
- Java provides the throw and throws keywords to explicitly raise an exception in the main() method.
- Java provides the finally block to ensure execution of cleanup code even when an exception occurs.