Exception handling

Exception Handling

- In Java, an exception is an event that disturbs the normal flow of the program. It is an object which is thrown at runtime.

- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

What is Exception Handling?

- Exception Handling is a mechanism to handle runtime errors

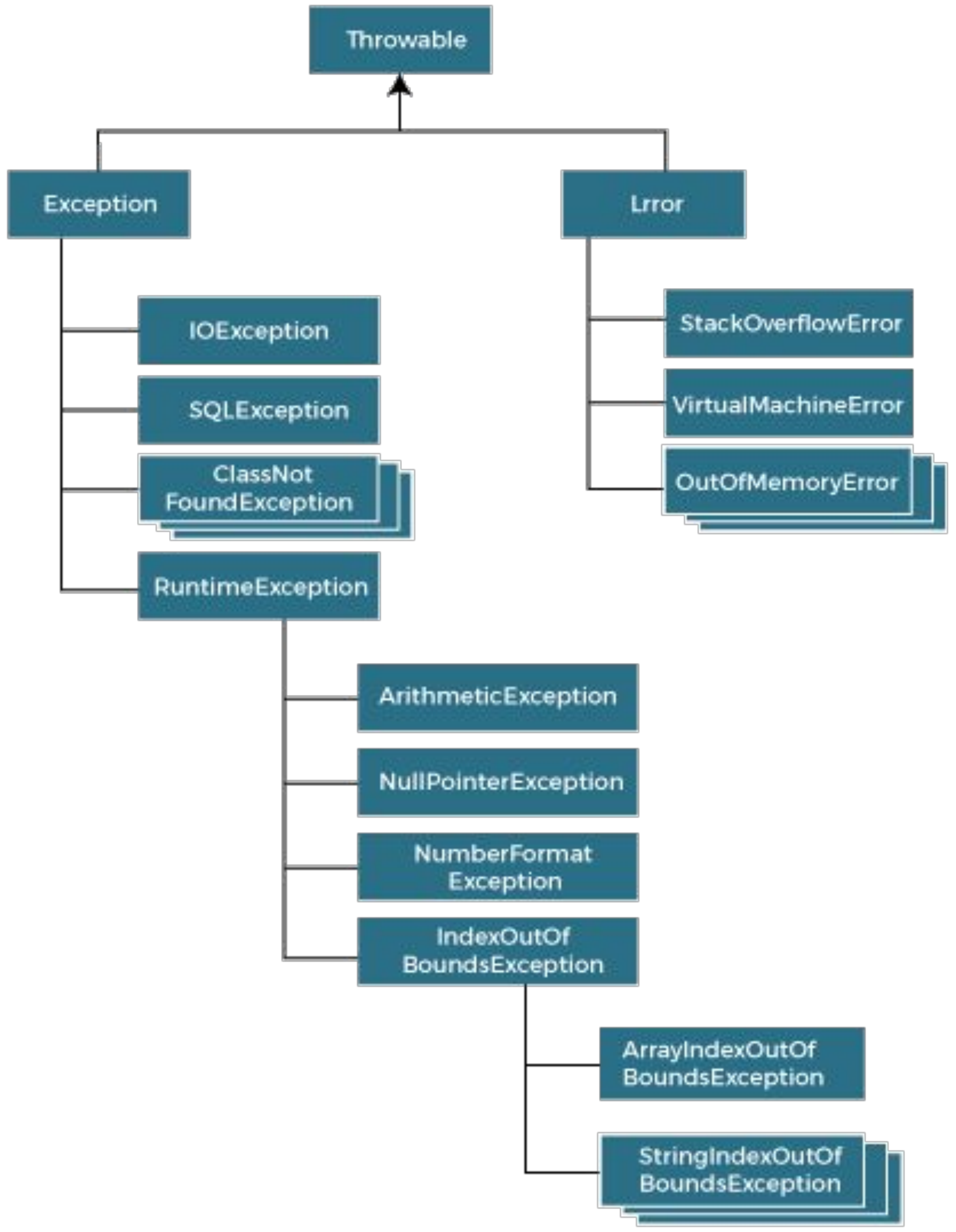
- Advantage of Exception Handling
- The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disturbs the normal flow of the application; that is why we need to handle exceptions.

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

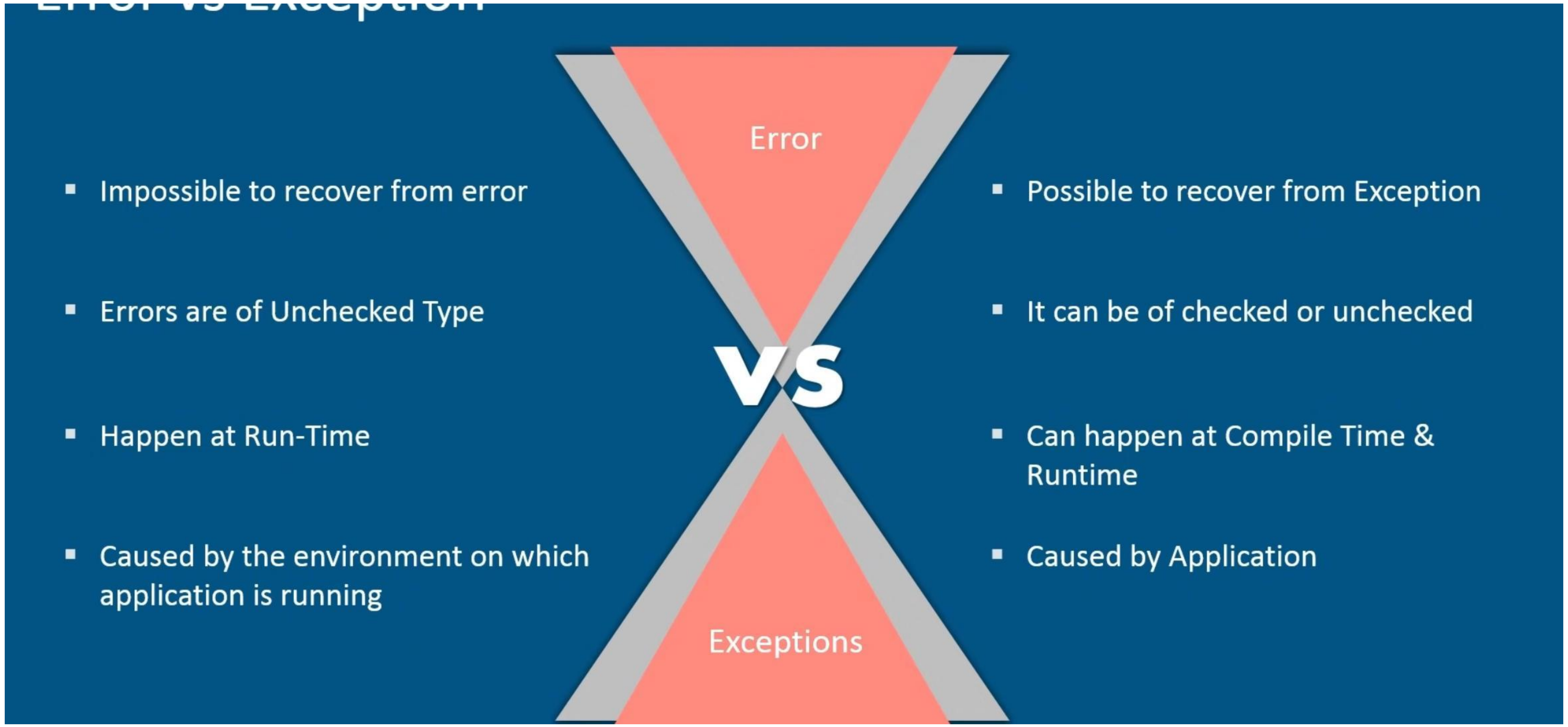
- Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

Hierarchy of Java Exception classes

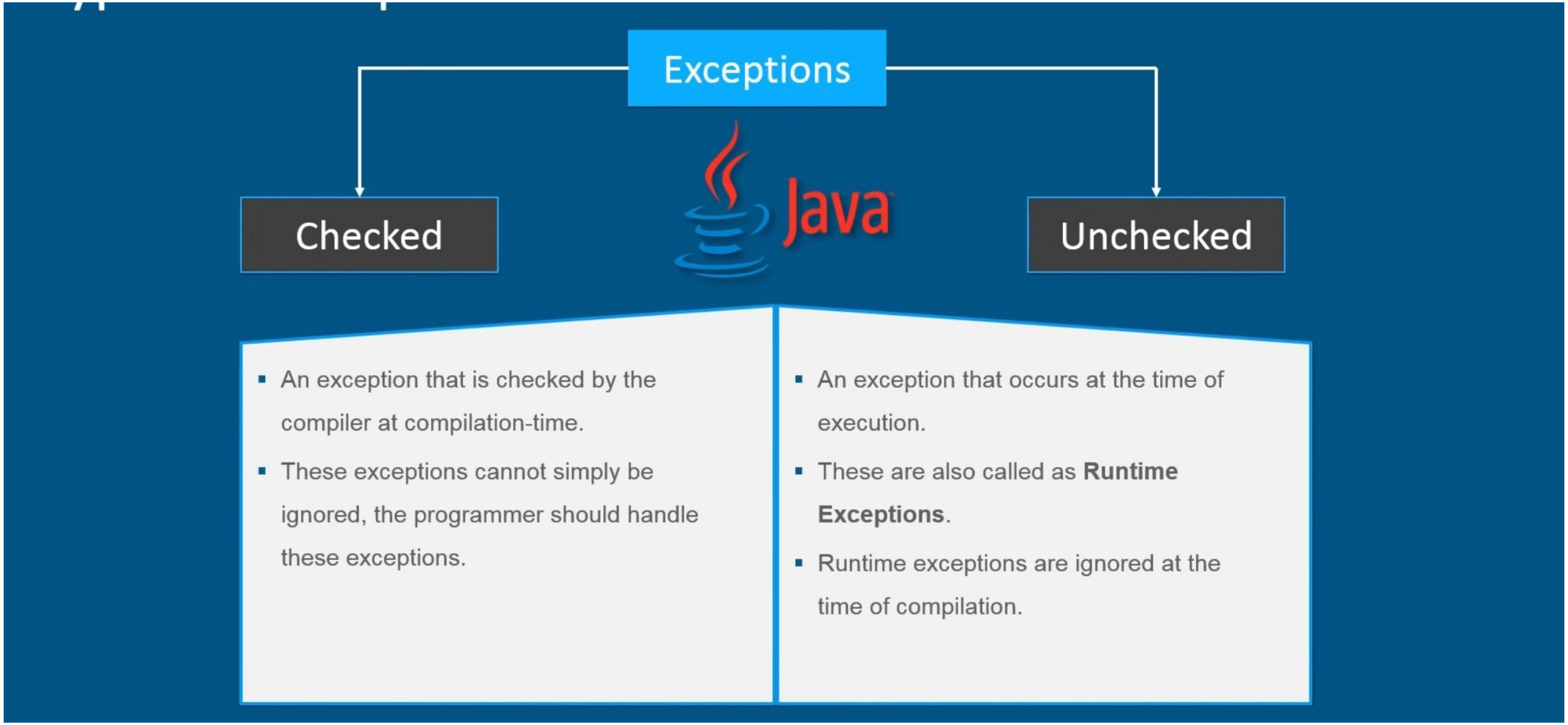
- The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error. The hierarchy of Java Exception classes is given below:


Throwable
Exception
Lrror
IOException
SQLException
ClassNot FoundException
RuntimeException
StackOverflowError
VirtualMachineError
OutOfMemoryError
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException

Errors and exception



Error	Exceptions
Impossible to recover from error	Possible to recover from Exception
Errors are of Unchecked Type	It can be of checked or unchecked
Happen at Run-Time	Can happen at Compile Time & Runtime
Caused by the environment on which application is running	Caused by Application



- An exception that is checked by the compiler at compilation-time.
- These exceptions cannot simply be ignored, the programmer should handle these exceptions.

- An exception that occurs at the time of execution.
- These are also called as **Runtime Exceptions**.
- Runtime exceptions are ignored at the time of compilation.

Basic Example format of Exception

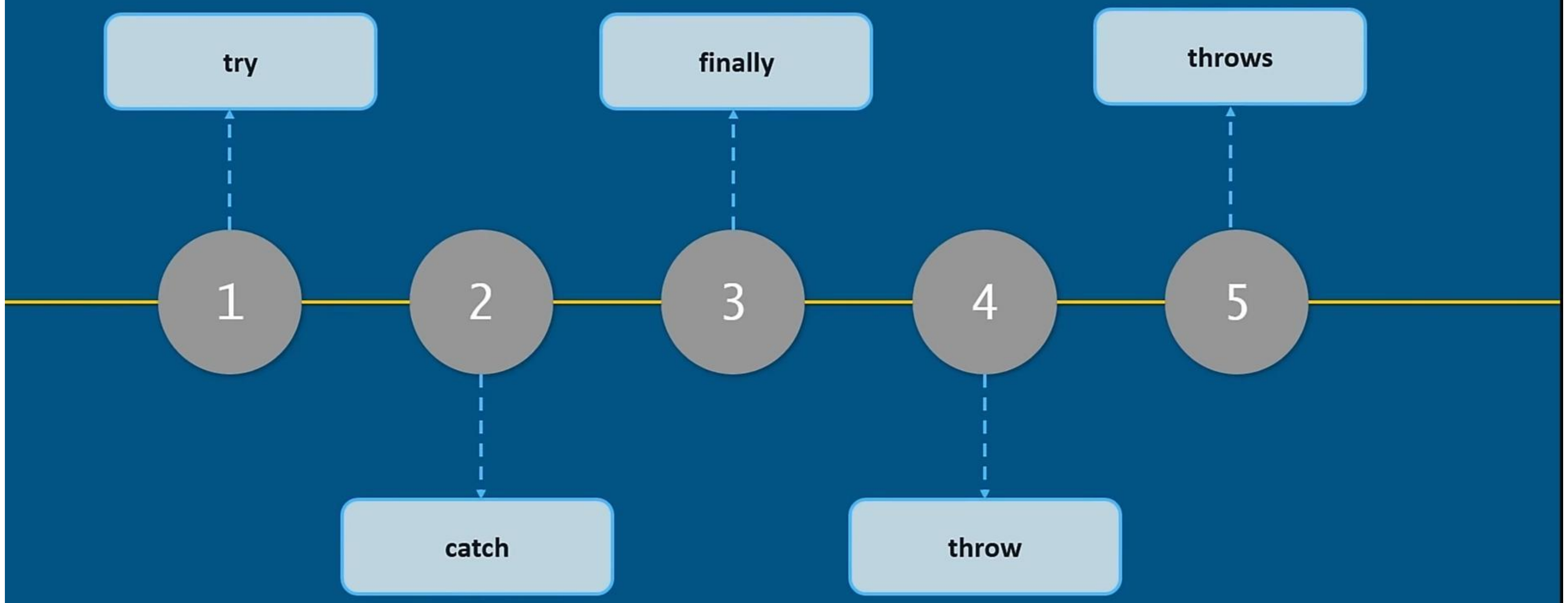
```
class Exception{
    public static void main(String args[]){
       try{
          //code that may raise exception
         }
       catch(Exception e){
          // rest of the program
         }
     }
}
```

Types of Java Exceptions

Built-in Exceptions

User Defined Exceptions

Exception Handling Methods



Exception Handling Methods

- **try**
- catch
- finally
- throw
- throws

Used to specify a block where we should place exception code.

Syntax:

```
try{
//code that throws exception
}catch(Exception_class_Name){}
```

Exception Handling Methods

- try
- catch
- finally
- throw
- throws

Used to handle the exception.

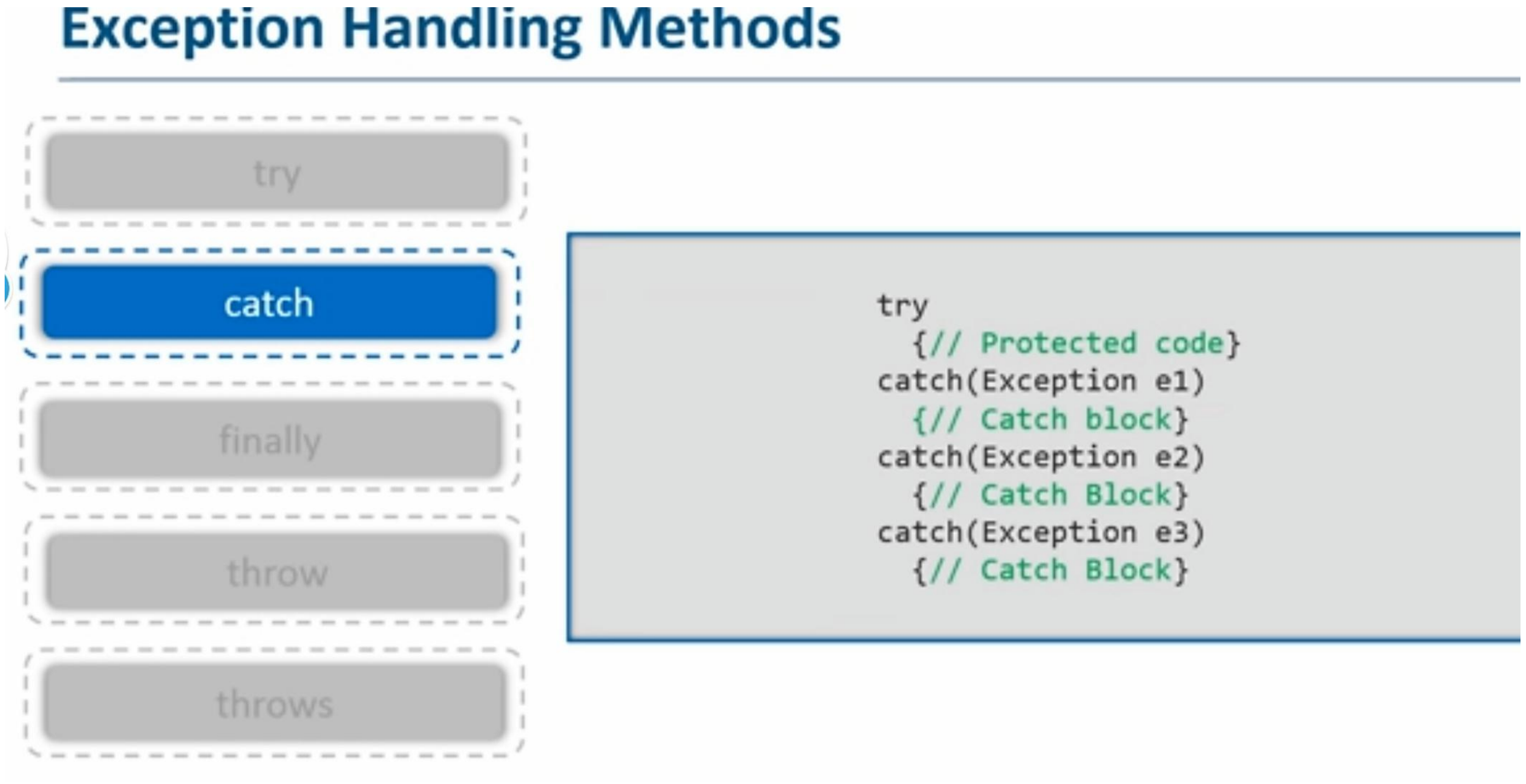
Syntax:

```
try {// Protected code}
catch(ExceptionName e1)
 // Catch block}
```

Nested try-catch

```
try
{
      try
      { Statements…
      }
       catch(exception e){}
       try
       {
       Statements
       }
        catch(exception e){}
}catch (exception e){}
```

Multiple catch blocks



```
try
  {// Protected code}
catch(Exception e1)
  {// Catch block}
catch(Exception e2)
  {// Catch Block}
catch(Exception e3)
  {// Catch Block}
```

```
Try
{
 statement1
Statement2
Statement3
}
```

Exception Handling Methods

try

catch

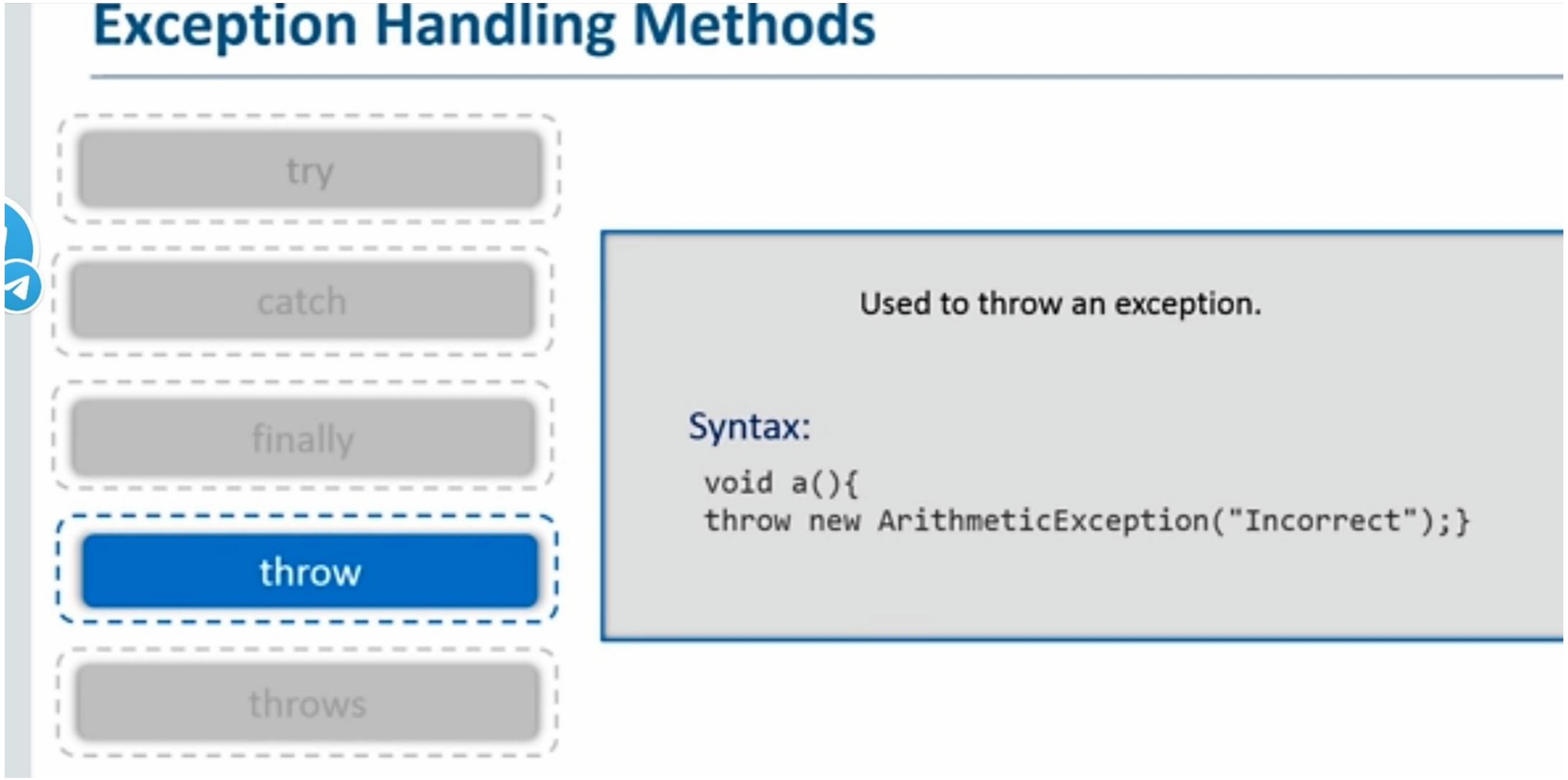
finally

throw

throws

Used to execute the important code of the program.

```
try
  {// Protected code}
catch(Exception e1)
  {// Catch block}
catch(Exception e2)
  {// Catch Block}
finally{
// This block is always executed
}
```



Exception Handling Methods

- try
- catch
- finally
- **throw**
- throws

Used to throw an exception.

Syntax:

```
void a(){
throw new ArithmeticException("Incorrect");}
```

throw

- The java throw keyword is used to throw an exception explicitly
- Specify exception object which is to be thrown
- We can also define our own set of conditions and throw an exception.
- The exception has some message with it that provides the error description.
- Syntax

throw new exception_class(“error message”);

e.g.

throw new IOException(“Device Error”);

Instance must be of type throwable or subclass of throwable.

Exception Handling Methods

try

catch

finally

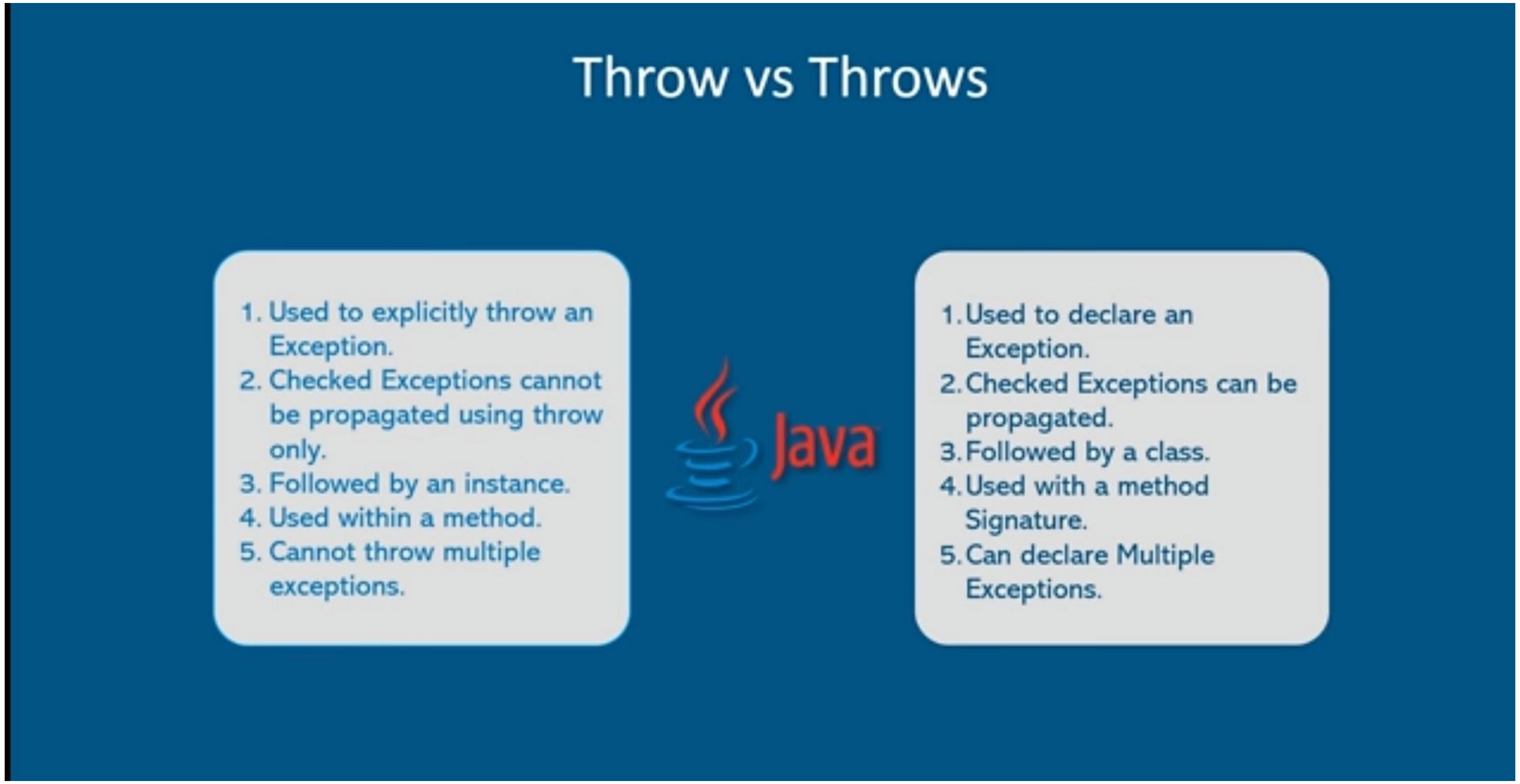
throw

throws

Used to declare exceptions.

Syntax:

```
void a()throws ArithmeticException {}
```



Throw vs Throws

1. Used to explicitly throw an Exception.
2. Checked Exceptions cannot be propagated using throw only.
3. Followed by an instance.
4. Used within a method.
5. Cannot throw multiple exceptions.

Java

1. Used to declare an Exception.
2. Checked Exceptions can be propagated.
3. Followed by a class.
4. Used with a method Signature.
5. Can declare Multiple Exceptions.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Java Exception Handling Example

```java
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
   }catch(ArithmeticException e){System.out.println(e);}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

- Common Scenarios of Java Exceptions
- There are given some scenarios where unchecked exceptions may occur. They are as follows:
- 1) A scenario where ArithmeticException occurs
- If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

- 2) A scenario where NullPointerException occurs
- If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

```
String s=null;
System.out.println(s.length());//NullPointerException
```

- 3) A scenario where NumberFormatException occurs
- If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a string variable that has characters; converting this variable into digit will cause NumberFormatException.

```
String s="abc";
1. int i=Integer.parseInt(s);//NumberFormatException
```

- 4) A scenario where ArrayIndexOutOfBoundsException occurs
- When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException. Consider the following statements.

```
1. int a[]=new int[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException
```

Custom Exceptions or User Defined exceptions

How to define

```
class Custom_exception_className extends Exception
{
  public String toString()
  {
    return " Exception message";
  }
}
```

-Use with throw and throws

```
import java.util.*;
class InvalidInputExp extends Exception
{
   public String toString()
   {
     return "invalid input";
}

public class ExceptionHandlingUserDefined
{
  static void validate(int number) throws InvalidInputExp
  {
   if(number>100)
   {
      throw new InvalidInputExp();
   }
  System.out.println("Valid Input");
  }
```

```
 public static void main(String args[])
 {
  Scanner s=new Scanner(System.in);
  System.out.println("Enter the number");
  int no=s.nextInt();
  try
     {
        validate(no);
     }
  catch(InvalidInputExp e)
     {
        System.out.println(e);
     }
 }
}
}
```

Exception Handling –Experiment no 6

- Try-catch block to handle divide by zero exception-arithmetic Exception c=a/b
- nested try-catch block to handle Array out of bounds exception
- Number format exception , null pointer exception using multiple catch
- Implement Throw ,throws and create custom Exception by adding validate() method to check valid input.
- Add finally block