

# Exception Handling

---

## Project Lombok

- Reference <https://projectlombok.org/>

## External system resources

- Following are the operating system resources that we can use for the application development:
  - Memory
  - Processor
  - Input and Output devices
  - File
  - Socket
  - Network Connections
  - Database connections
  - Operating System API
- In the context of Java, all above resources are non Java resources. These are also called as unmanaged resources(except memory).
- In the context of Java, resource is any external system resource that we can use in the application.
- Since operating system resources are limited, we should use it carefully.

## Exception Concept

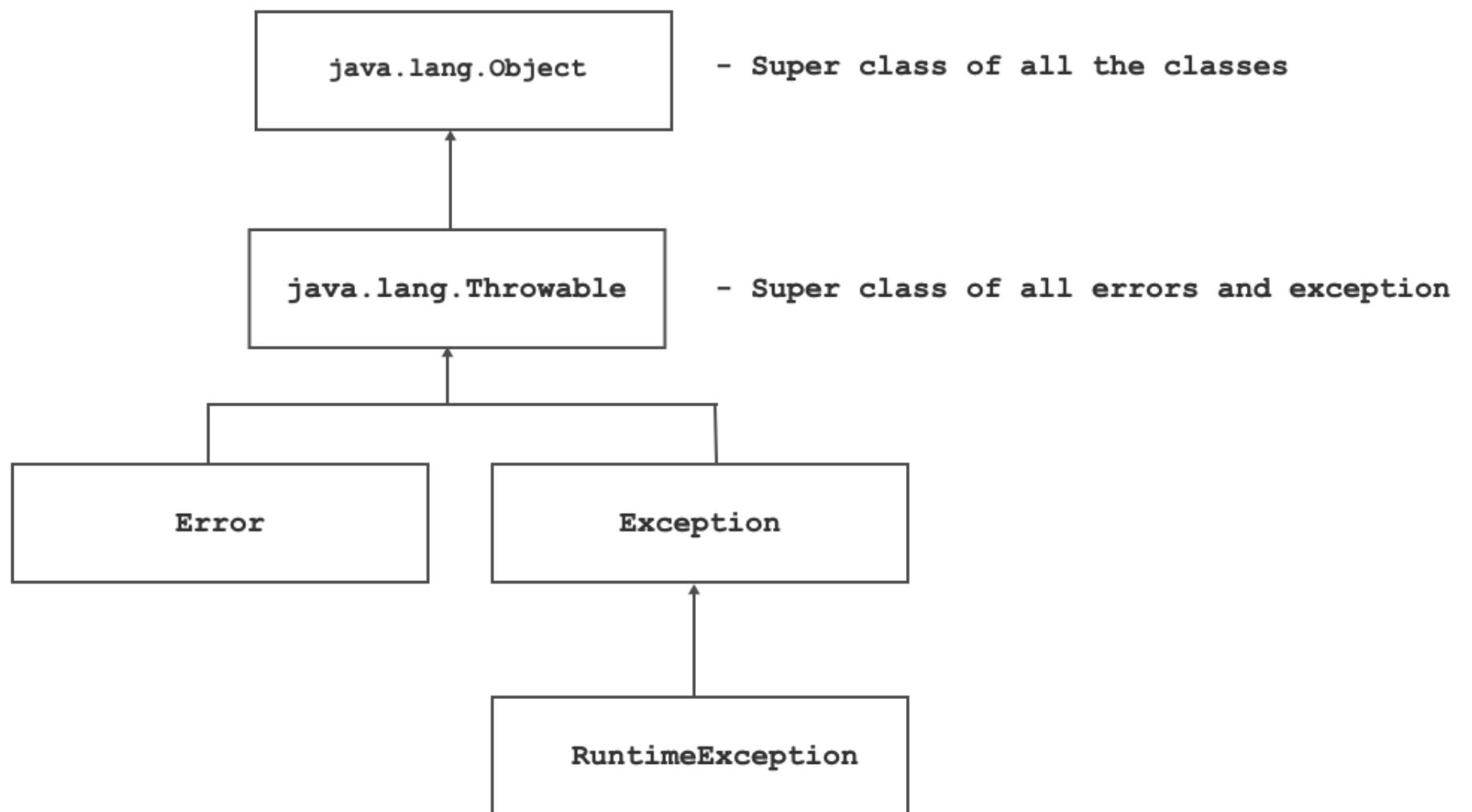
- Definition
  - Exception is an issue / unexpected event / instance which occurs during execution of application.
  - Exception is an instance which is used to acknowledge user of the system if any exeption situation occurs in the code.
  - If we want to manages OS resources carefully then we should use exception handling in Java.

## Throwable class Hierarchy

- java.lang.Object is ultimate super class of all the classes in Java language.
- Methods of java.lang.Object class:
  - public String toString( );
  - public boolean equals( Object obj );
  - public native int hashCode();
  - protected native Object clone( )throws CloneNotSupportedException
  - protected void finalize( )throws Throwable;
  - public final native Class<?> getClass();
  - public final void wait( )throws InterruptedException
  - public final native void wait( long timeOut )throws InterruptedException
  - public final void wait( long timeOut, int nanos)throws InterruptedException
  - public final native notify( );
  - public final native notifyAll( );

**java.lang.Throwable:**

## Object Oriented Programming with Java



- The Throwable class is the superclass of all errors and exceptions in the Java language.
- Only objects that are instances of Throwable class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.
- Consider code in C++

```
int main( void ){
    int num1;
    cout<<"Enter number : ";
    cin>>num1;

    int num2;
    cout<<"Enter number : ";
    cin>>num2;
    try{
        if( num2 == 0 ){
            //throw 0; //OK
            //throw ArithmeticException( "Divide by zero exception"); //OK
            throw "Divide by zero exception"; //OK
        }else
            int result = num1 / num2;
            cout<<"Result : "<<result<<endl;
        }
    }catch( string &ex ){
        cout<<ex<<endl;
    }
    return 0;
}
```

- Consider code in Java

```
class MyException{
    private String message;
    public MyException(String message) {
        this.message = message;
    }
}

public class Program {
    public static void main(String[] args){
        int num1 = 10;
        int num2 = 0;
        try {
            if( num2 == 0 )
                //throw 0; //No exception of type int can be thrown; an exception type must be a subclass of
```

## Object Oriented Programming with Java

```
Throwable
    //throw "/" by 0"; //No exception of type String can be thrown; an exception type must be a subclass
of Throwable
    throw new MyException("/ by 0"); //No exception of type MyException can be thrown; an exception type
must be a subclass of Throwable
    int result = num1 / num2;
    System.out.println("Result : "+result);
} catch( Exception ex ) {
    //TODO
}
}
```

- Consider following code:

```
public class Program {
    public static void main(String[] args){
        int num1 = 10;
        int num2 = 0;
        try {
            if( num2 == 0 ) {
                ArithmeticException ex = new ArithmeticException("Value of denominator should not be zero");
                throw ex; //OK
            }
            int result = num1 / num2;
            System.out.println("Result : "+result);
        } catch( Exception ex ) {

        }

    }
}
```

- Similarly, only Throwable class or one of its subclasses can be the argument type in a catch clause.
- Consider following code:

```
public class Program {
    public static void main(String[] args) {

        String str = null;
        str.charAt(10);
        int num1 = 10;
        int num2 = 0;
        try {
            int result = num1 / num2;
            System.out.println("Result : " + result);
        } catch (ArithmeticException ex) {    //No exception of type String can be thrown; an exception type
must be a subclass of Throwable

        }
    }
}
```

- Consider following code:

```
public class Program {
    public static void main(String[] args) {

        String str = null;
        str.charAt(10);
        int num1 = 10;
        int num2 = 0;
        try {
            int result = num1 / num2;
            System.out.println("Result : " + result);
        } catch (ArithmeticException ex) {    //
//TODO
        }
    }
}
```

- Constructor Summary

- public Throwable()

```
Throwable tw = new Throwable();
```

- public Throwable(String message)

```
String message = "error message";
Throwable tw = new Throwable( message );
//or
Throwable tw = new Throwable( "error message" );
```

- public Throwable(Throwable cause)

```
Throwable cause = new Throwable( "error message" );
Throwable tw = new Throwable( cause );
//or
Throwable tw = new Throwable( new Throwable( "error message" ) );
```

- public Throwable(String message, Throwable cause)

```
Throwable cause = new Throwable( "error message" );
Throwable tw = new Throwable( "new error message", cause );
```

- protected Throwable(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace)

```
Throwable cause = new Throwable( "error message" );
Throwable tw = new Throwable( "new error message", cause, false, true );
```

- Method Summary

- public String getMessage()
- public Throwable initCause(Throwable cause)
- public Throwable getCause()
- public final void addSuppressed(Throwable exception)
- public void printStackTrace()
- public void printStackTrace(PrintStream s)
- public void printStackTrace(PrintWriter s)
- public StackTraceElement[] getStackTrace()

### Error versus Exception

- Error and Exception are direct sub classes of java.lang.Throwable class.

### Error

- An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch.
- Most such errors are abnormal conditions.
- Runtime error which gets generated due to environmental condition( hardware failure / OS failure / JVM failure etc ) is considered as error in java programming language.
- Consider following code:

```
public class Program {
    public static void main(String[] args) {
        try {
            int[] arr = new int[ Integer.MAX_VALUE ];
            System.out.println( Arrays.toString(arr));
        } catch( OutOfMemoryError error ) {
            System.out.println(error.getMessage()); //Requested array size exceeds VM limit
        }
    }
}
```

- We can write try-catch block to handle errors. But we can not recover from errors hence it is not recommended to use try catch block for the errors.
- Example:
  - OutOfMemoryError
  - StackOverflowError
  - VirtualMachineError

### Exception

- The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.
- Runtime error which gets generated due to application is considered as exception in java programming language.
- We can use try-catch block to handle exception.
- Example:
  - CloneNotSupportedException
  - InterruptedException
  - NumberFormatException
  - NullPointerException
  - NegativeArraySizeException
  - ArrayIndexOutOfBoundsException
  - ArrayStoreException
  - ClassCastException
  - ArithmeticException

### Checked versus unchecked exception

- Checked exception and unchecked exception are types of exception in Java, which are designed for Java compiler( Not for JVM ).

### Unchecked Exception

- java.lang.RuntimeException is considered as super class of all the unchecked exception.
- java.lang.RuntimeException and all its sub classes are considered as unchecked exception.
- Examples of unchecked exception
  - RuntimeException
  - NumberFormatException
  - NullPointerException
  - NegativeArraySizeException
  - ArrayIndexOutOfBoundsException
  - ArrayStoreException
  - ClassCastException
  - ArithmeticException
- Compiler do not force developer to handle or to use try-catch block for unchecked exception.

### Checked Exception

- java.lang.Exception is considered as super class of all the checked exception.
- java.lang.Exception and all its sub classes except java.lang.RuntimeException(and its sub classes ) are considered as checked exceptions
- Examples of checked exception
  - java.lang.CloneNotSupportedException
  - java.lang.InterruptedException
  - java.io.IOException
  - java.sql.SQLException
- Compiler force developer to handle or to use try-catch block for checked exception.

### AutoCloseable and Closeable

- Closeable is an interface which is declared in java.io package.
  - Method: void close() throws IOException
- It is introduced in JDK 1.5
- Consider following code:

```
import java.io.Closeable;
import java.io.IOException;
import java.util.Scanner;

class Test implements Closeable{
    private Scanner sc;
    public Test() {
        this.sc = new Scanner(System.in);
    }
}
```

```
}

@Override
public void close() throws IOException {
    this.sc.close();
}

}

public class Program {
    public static void main(String[] args) {
        try {
            Test t = new Test();

            t.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

- "void close() throws IOException" is a method of java.io.Closeable interface which is used to clean/release resources.
- If any class implements Closeable interface then that class has a ability to close resources using close method.
- AutoCloseable is an interface which is declared in java.lang package.
  - Method: void close()throws Exception
- It is introduced in JDK 1.7
- AutoCloseable is same as Closeable with gurantee of calling close method automatically.

```
//Class Test => Resource Type
class Test implements AutoCloseable{
    private    sc;
    public Test() {
        this.sc = new Scanner(System.in);
    }

    @Override
    public void close() throws Exception {
        this.sc.close();
    }
}

public class Program {
    public static void main(String[] args) {
        try {
            Test t = new Test(); //new Test()  => Resource

            t.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

- Any class which implements AutoCloseable / Closeable interface is called as resource type and its instance is called resource.
- If we use try with resource then close() method gets called automatically.

### Exception handling using try catch throw throws and finally

- If we want to handle exception then we should use 5 keywords in java:
  - try
  - catch
  - throw
  - throws
  - finally
- While performing arithmetic operation, if we get any exception condition like "divide by zero" then JVM throws ArithmeticException.

**try**

## Object Oriented Programming with Java

- It is a keyword in Java.
- If we want to keep watch on single statement or group of statements for exception then we should use try block / handler.
- we can not define try block after catch/finally block.
- Try block must have at least one catch block or finally block or resource statement.
- Consider following syntax:

```
public static void main(String[] args) {  
    try{  
        //TODO  
    }catch( Exception ex ){  
        //TODO  
    }  
}
```

```
public static void main(String[] args) {  
    try{  
        //TODO  
    }finally{  
        //TODO  
    }  
}
```

```
public static void main(String[] args) {  
    try(Scanner sc = new Scanner()){ //try-with-resource  
        //TODO  
    }  
}
```

### throw

- It is a keyword in java.
- If we want to generate new exception then we should use throw keyword.
- Only objects that are instances of Throwable class (or one of its subclasses) are thrown by the JVM or can be thrown by the Java throw statement.

```
String ex = new String("Divide by zero exception");  
throw ex;//No exception of type String can be thrown; an exception type must be a subclass of Throwable
```

- throw statement is a jump statement.

### catch

- It is a keyword in Java.
- To handle exception, we should use catch block / handler.
- We can not define catch block before try and after finally block.
- Catch block can handle exception thrown from try block only.
- For single try block, we can provide multiple catch block. In this case, JVM can execute only one catch depending on the situation.
- In a single catch block, we can handle multiple specific exception.

```
catch ( ArithmeticException | InputMismatchException ex ) {    //Multi catch block  
    ex.printStackTrace();  
}
```

- Do not Repeat Yourself( DRY ).
- If we want to handle multiple exceptions of super and sub types then first we must handle sub types exception.

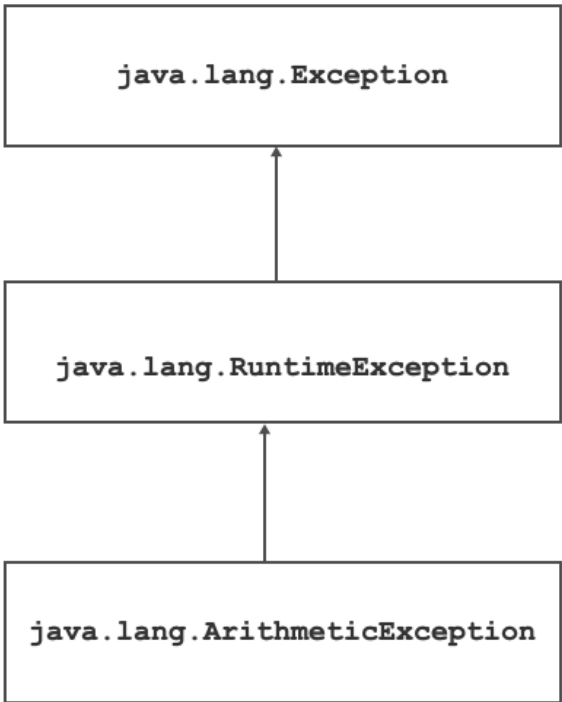
```
public static void main(String[] args) {  
    Scanner sc = null;  
    try {  
        System.out.println("Opening resource");  
    }  
}
```

```
sc = new Scanner(System.in);

System.out.print("Num1    :  ");
int num1 = sc.nextInt();

System.out.print("Num2    :  ");
int num2 = sc.nextInt();

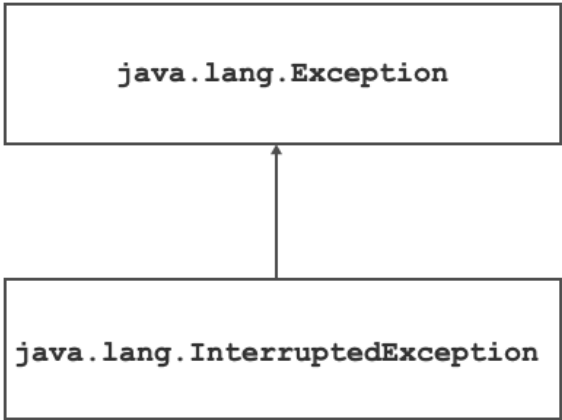
if( num2 == 0 )
    throw new ArithmeticException("Divide by zero exception");
int result = num1 / num2;
System.out.println("Result    :  "+result);
}catch ( ArithmeticException  ex) {
    ex.printStackTrace();
}catch ( RuntimeException  ex) {
    ex.printStackTrace();
}catch ( Exception  ex) {
    ex.printStackTrace();
}
}
```



```
ArithmeticException ex = new ArithmeticException();

RuntimeException ex = new ArithmeticException();//Upcasting

Exception ex = new ArithmeticException(); //Upcasting
```



```
InterruptedException ex = new InterruptedException(); // OK

Exception ex = new InterruptedException(); // Upcasting
```

- Using java.lang.Exception class we can define catch block which can handle any checked as well as unchecked exception.

```
try{
    //TODO
}catch( Exception ex ){ //Generic catch block
    ex.printStackTrace();
}
```

- Generally, generic catch block comes after all specific catch blocks.

finally

- It is a keyword in Java.
- If we want to close or release local resources then we should use finally block.



## Object Oriented Programming with Java

- For given try block we can provide only one finally block.
- We can define block after all try and catch blocks.
- JVM always execute finally block.

try with resource

```
public static void main(String[] args) {
    //try ( Program p = new Program()) {    //Not Ok: The resource type Program does not implement
    java.lang.AutoCloseable
    try( Scanner sc = new Scanner(System.in)){
        System.out.print("Num1 : ");
        int num1 = sc.nextInt();

        System.out.print("Num2 : ");
        int num2 = sc.nextInt();

        if( num2 == 0 )
            throw new ArithmeticException("Divide by zero exception");
        int result = num1 / num2;
        System.out.println("Result : "+result);

    }catch ( Exception ex) {
        ex.printStackTrace();
    }
}
```

throws

- It is a keyword in Java
- If we want delegate exception from method to the caller method then we should use throws keyword/clause.

```
public class Program {
    public static void displayRecord( ) throws InterruptedException {
        for( int count = 1; count <= 10; ++ count ) {
            System.out.println("Count : "+count);
            Thread.sleep(500);
        }
    }
    public static void main(String[] args) {
        try {
            Program.displayRecord();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Custom exception and its need.

- JVM do not understand exceptional conditions in the business logic. To handle it we should define user defined / custom exception class.
- How to define custom unchecked exception class?

```
class StackOverflowException extends RuntimeException{
    //TODO
}
```

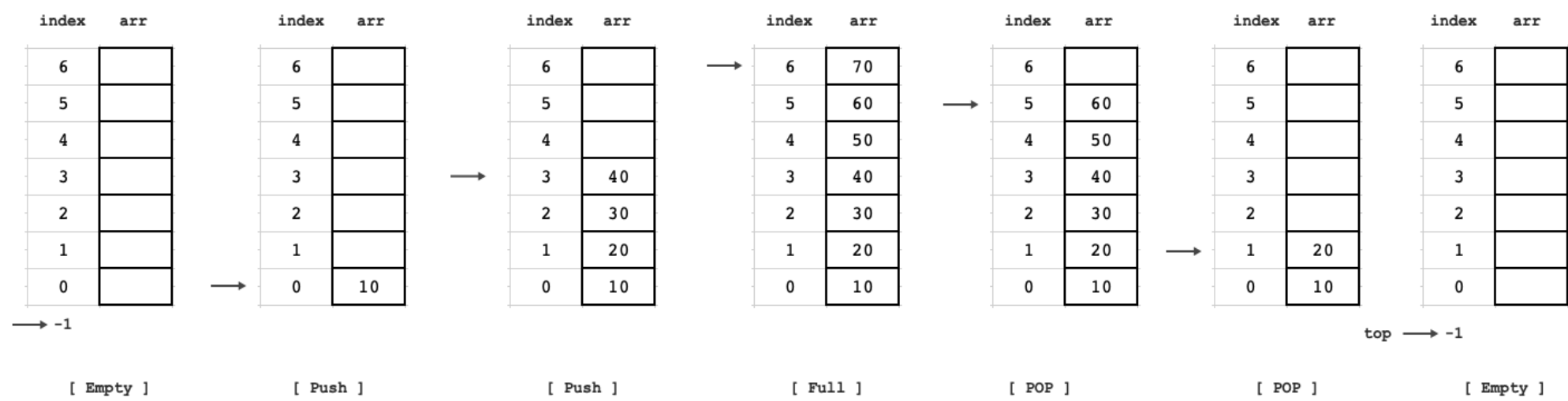
- How to define custom checked exception class?

```
class StackOverflowException extends Exception{
    //TODO
}
```

Exception chaining

Object Oriented Programming with Java

Stack: Last In First Out (LIFO) operations



- Process of handling exception by throwing new type of exception is called as exception chaining.
- Consider following code:

```
package org.example;
abstract class A{
    public abstract void print( );
}
class B extends A{
    @Override
    public void print() throws RuntimeException{
        try {
            for( int count = 1; count <= 10; ++ count ) {
                System.out.println("Count : "+count);
                Thread.sleep(250);
            }
        } catch (InterruptedException cause) {
            throw new RuntimeException(cause); //Exception Chaining
        }
    }
}
public class Program {
    public static void main(String[] args) {
        try {
            A a = new B();
            a.print();//Dynamic method dispatch
        } catch (RuntimeException e) {
            //e.printStackTrace();
            Throwable cause = e.getCause();
            System.out.println(cause);
        }
    }
}
```