Try-with-resources

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

class TryWithResources {
      public static void main(String []args){
            System.out.println("In Main method .....");
            new TryWithResources().openFile();
            System.out.println("Existing Main Method .....");
      }
    public void openFile(){
        try(FileReader reader = new
FileReader("/home/neha/java.info/testFile")) {
            int i=0;
            while(i != -1){
                //reader.read() may throw IOException
                i = reader.read();
                System.out.print((char) i );
            }
            System.out.println("--- File End ---");
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

The FileInputStream variable is declared inside the parentheses after the try keyword. Additionally, a FileInputStream is instantiated and assigned to the variable.
When the try block finishes the FileInputStream will be closed automatically. This is possible because FileInputStream implements the Java interface java.lang.AutoCloseable. All classes implementing this interface can be used inside the try-with-resources construct.

Try-with-multiple-resources

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;

class TryWithMultipleResources {
      public static void main(String []args){
            System.out.println("In Main method .....");
            new TryWithMultipleResources().openFile();
            System.out.println("Existing Main Method .....");
      }
    public void openFile(){
        try(FileReader inputStream = new
FileReader("/home/neha/java.info/testFile");
                  BufferedReader reader = new BufferedReader(inputStream)
            ){
            String inputLine = reader.readLine();
            while(inputLine!= null){
                  System.out.println(inputLine);
                  inputLine = reader.readLine();
            }
            System.out.println("--- File End ---");
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

OUTPUT:

```
      In Main method .....
      https://github.com/expressjs/express-paginate
      https://www.npmjs.com/package/pagination
      https://programmerblog.net/nodejs-mysql-pagination-example-beginners/

      --- File End ---
      Existing Main Method .....
```

An FileReader and a BufferedReader two resources inside the parentheses after the try keyword. Both of these resources will be closed automatically when execution leaves the try block.

The resources will be closed in reverse order of the order in which they are created / listed inside the parentheses. First the BufferedReader will be closed, then the FileReader.

How actually it works?

In java 7, we have a new super interface java.lang.AutoCloseable. This interface have one method:

void close() throws Exception;

Java docs recommend this interface to be implemented on any resource that must be closed when it is no longer needed.

When we open any such AutoCloseable resource in special try-with-resource block, immediately after finishing the try block, JVM calls this close() method on all resources initialized in “try()” block.

For example, BufferedReader has implemented close() method file this:

```
public void close() throws IOException {
    synchronized (lock) {
        if (in == null)
            return;
        in.close();
        in = null;
        cb = null;
    }
}
```

Custom AutoClosable Implementations

```
class CustomAutoClosable {
      public static void main(String[] args)
    {
        try(CustomizedResource object = new CustomizedResource())
        {
            object.accessingResource();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
class CustomizedResource implements AutoCloseable
{
    public void accessingResource() {
        System.out.println("Accessing the resource");
    }

    @Override
    public void close() throws Exception {
        System.out.println("CustomizedResource closed automatically");
    }
}
```

```
OUTPUT:
      Accessing the resource
      CustomizedResource closed automatically
```

SUMMING UP

Before java 7, we had to use finally blocks to cleanup the resources.

With java 7, no need to explicit resource cleanup. It's done automatically.

Cleanup happens because of new interface AutoCloseable. Its close method is invoked by JVM as soon as try block finishes.

If you want to use this in custom resources, then implementing AutoCloseable interface is mandatory. otherwise program will not compile.

You are not supposed to call close() method in your code. This should be called automatically by JVM. Calling it manually may cause unexpected results.