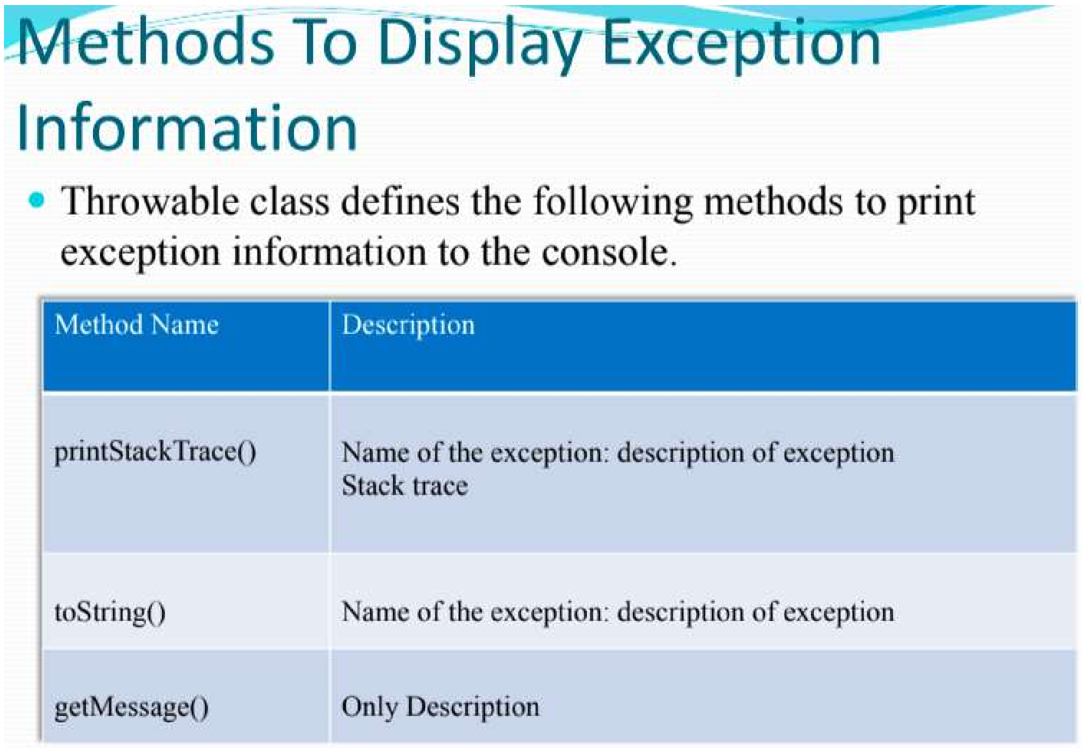
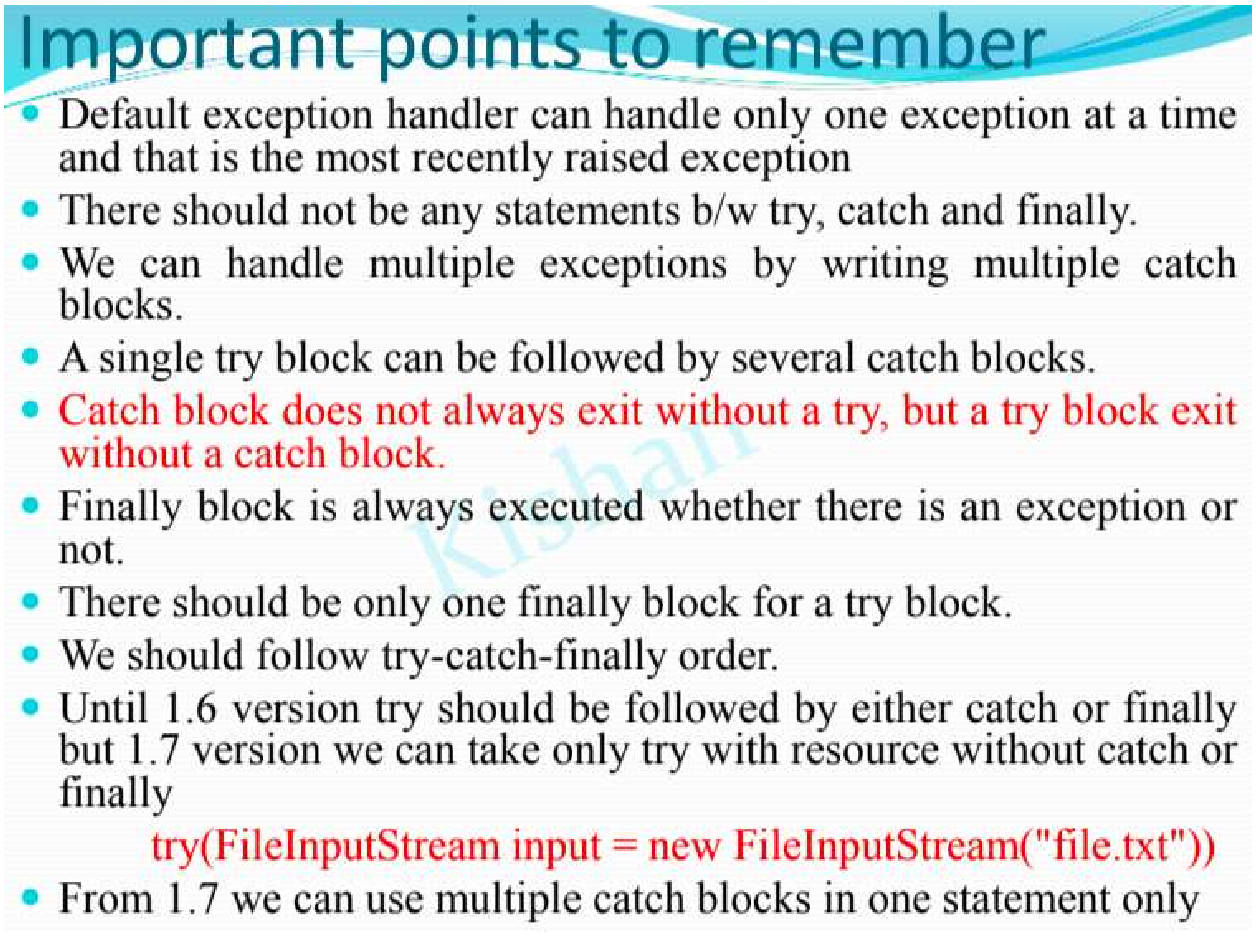


Methods To Display Exception Information

- Throwable class defines the following methods to print exception information to the console.

Method Name	Description
printStackTrace()	Name of the exception: description of exception Stack trace
toString()	Name of the exception: description of exception
getMessage()	Only Description

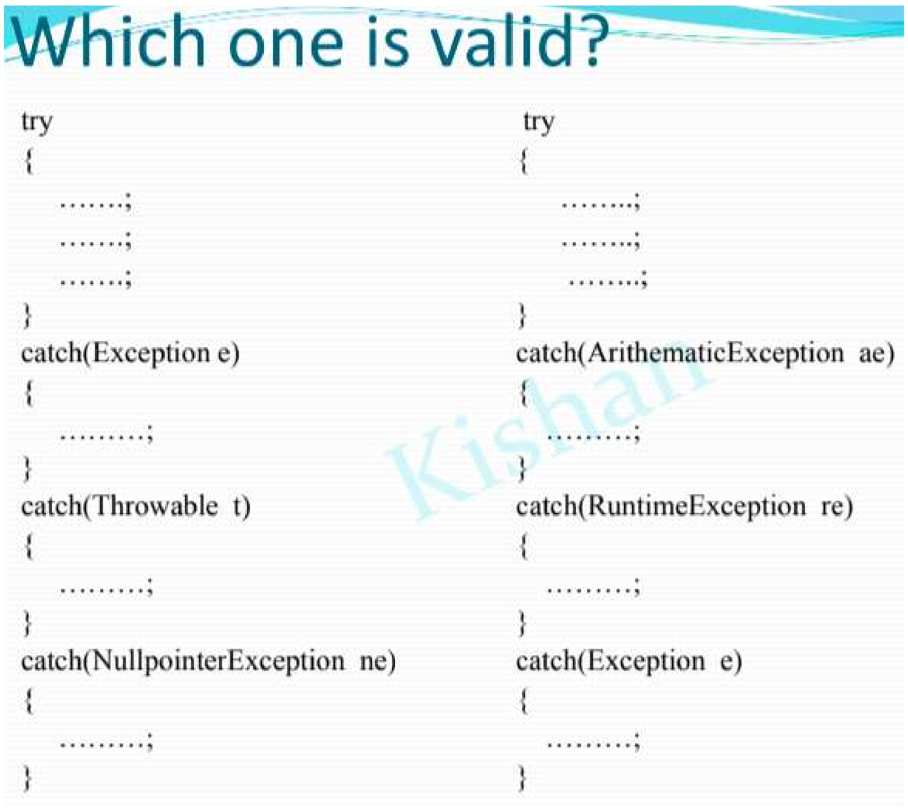


Important points to remember

- Default exception handler can handle only one exception at a time and that is the most recently raised exception
- There should not be any statements b/w try, catch and finally.
- We can handle multiple exceptions by writing multiple catch blocks.
- A single try block can be followed by several catch blocks.
- Catch block does not always exit without a try, but a try block exit without a catch block.
- Finally block is always executed whether there is an exception or not.
- There should be only one finally block for a try block.
- We should follow try-catch-finally order.
- Until 1.6 version try should be followed by either catch or finally but 1.7 version we can take only try with resource without catch or finally

```
try(FileInputStream input = new FileInputStream("file.txt"))
```

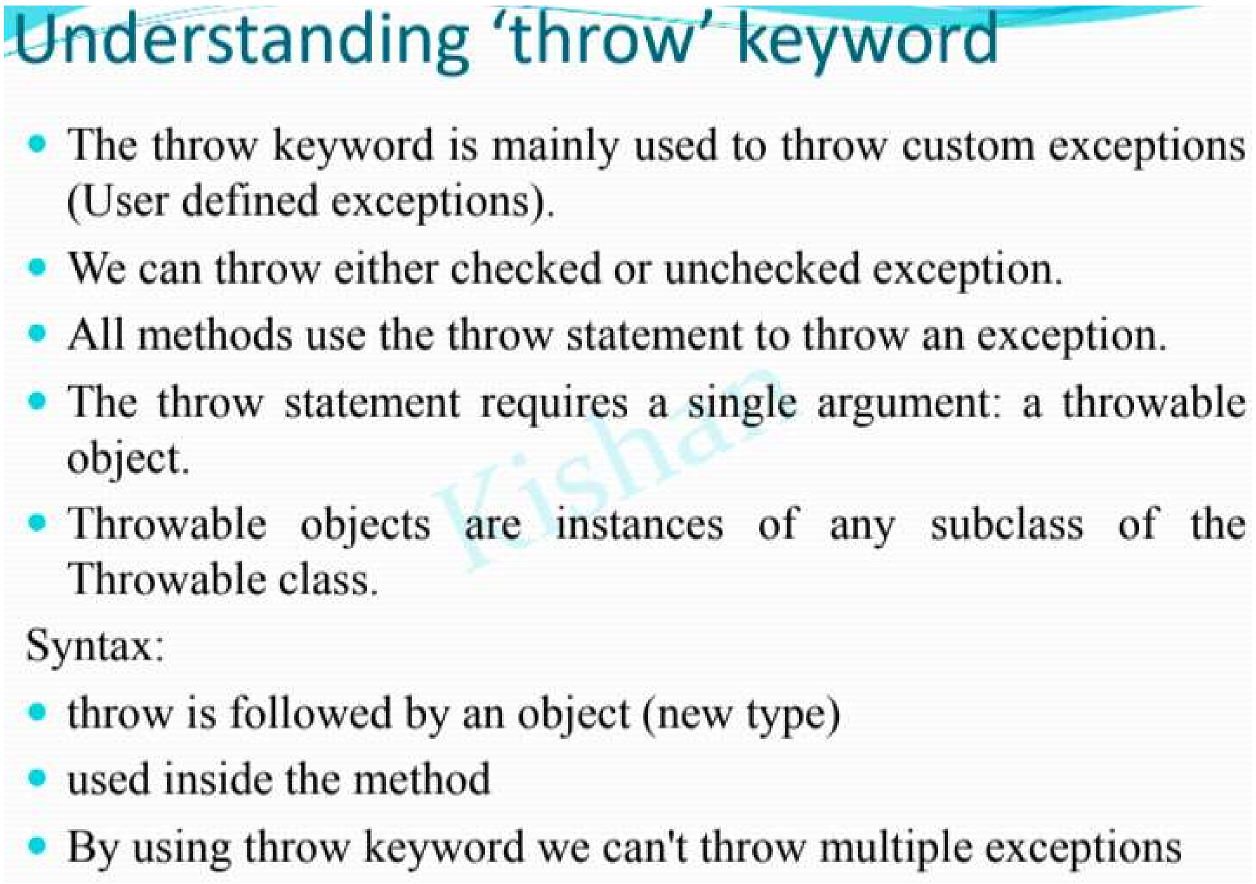
- From 1.7 we can use multiple catch blocks in one statement only



Which one is valid?

```
try
{
    .......;
    .......;
    .......;
}
catch(Exception e)
{
    .........;
}
catch(Throwable  t)
{
    .........;
}
catch(NullpointerException  ne)
{
    .........;
}
```

```
try
{
     ........;
     ........;
      ........;
}
catch(ArithematicException  ae)
{
    .........;
}
catch(RuntimeException  re)
{
    .........;
}
catch(Exception  e)
{
    .........;
}
```

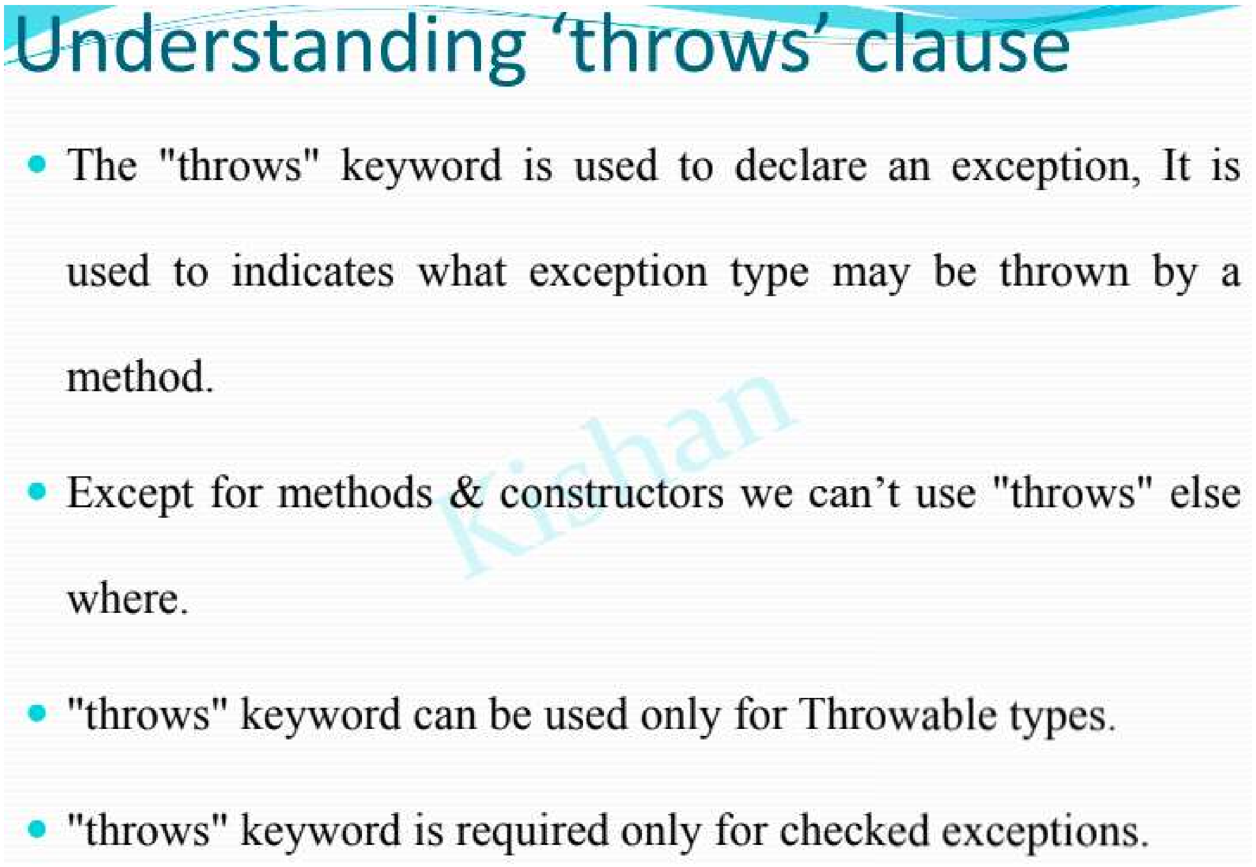


Understanding 'throw' keyword

- The throw keyword is mainly used to throw custom exceptions (User defined exceptions).
- We can throw either checked or unchecked exception.
- All methods use the throw statement to throw an exception.
- The throw statement requires a single argument: a throwable object.
- Throwable objects are instances of any subclass of the Throwable class.

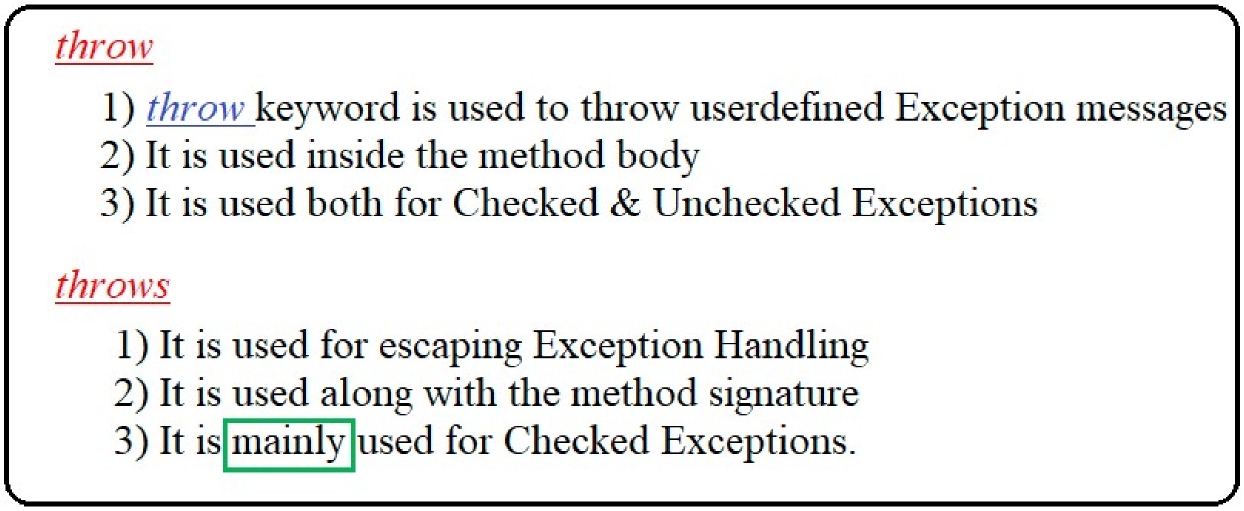
Syntax:

- throw is followed by an object (new type)
- used inside the method
- By using throw keyword we can't throw multiple exceptions



Understanding 'throws' clause

- The "throws" keyword is used to declare an exception, It is used to indicates what exception type may be thrown by a method.
- Except for methods & constructors we can't use "throws" else where.
- "throws" keyword can be used only for Throwable types.
- "throws" keyword is required only for checked exceptions.

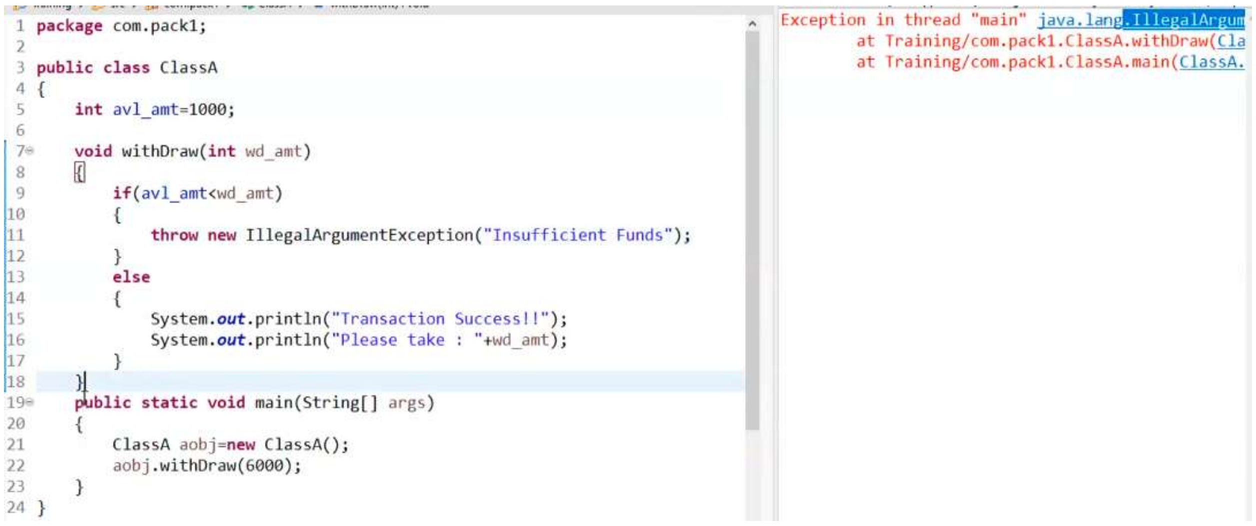


throw

1) *throw* keyword is used to throw userdefined Exception messages
2) It is used inside the method body
3) It is used both for Checked & Unchecked Exceptions

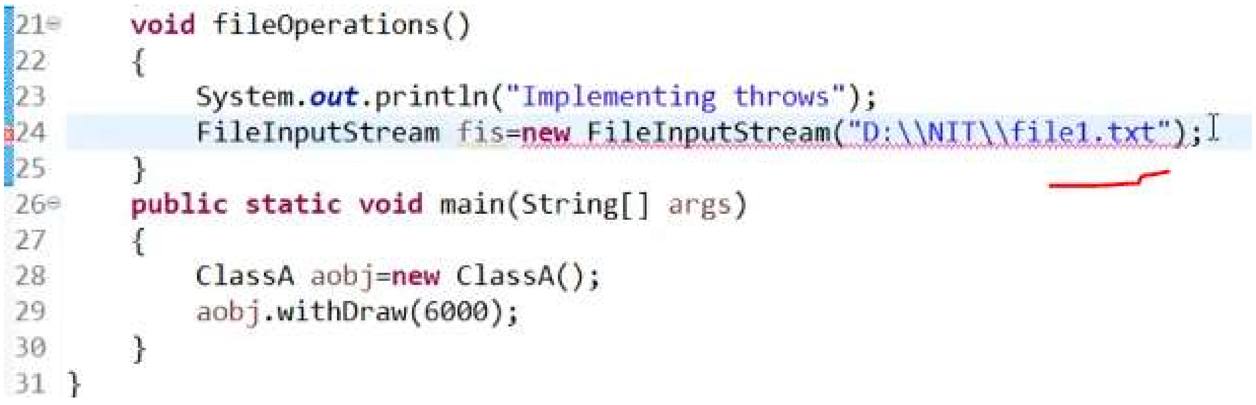
throws

1) It is used for escaping Exception Handling
2) It is used along with the method signature
3) It is mainly used for Checked Exceptions.



```
package com.pack1;

public class ClassA
{
    int avl_amt=1000;

    void withDraw(int wd_amt)
    {
        if(avl_amt<wd_amt)
        {
            throw new IllegalArgumentException("Insufficient Funds");
        }
        else
        {
            System.out.println("Transaction Success!!");
            System.out.println("Please take : "+wd_amt);
        }
    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        aobj.withDraw(6000);
    }
}
```

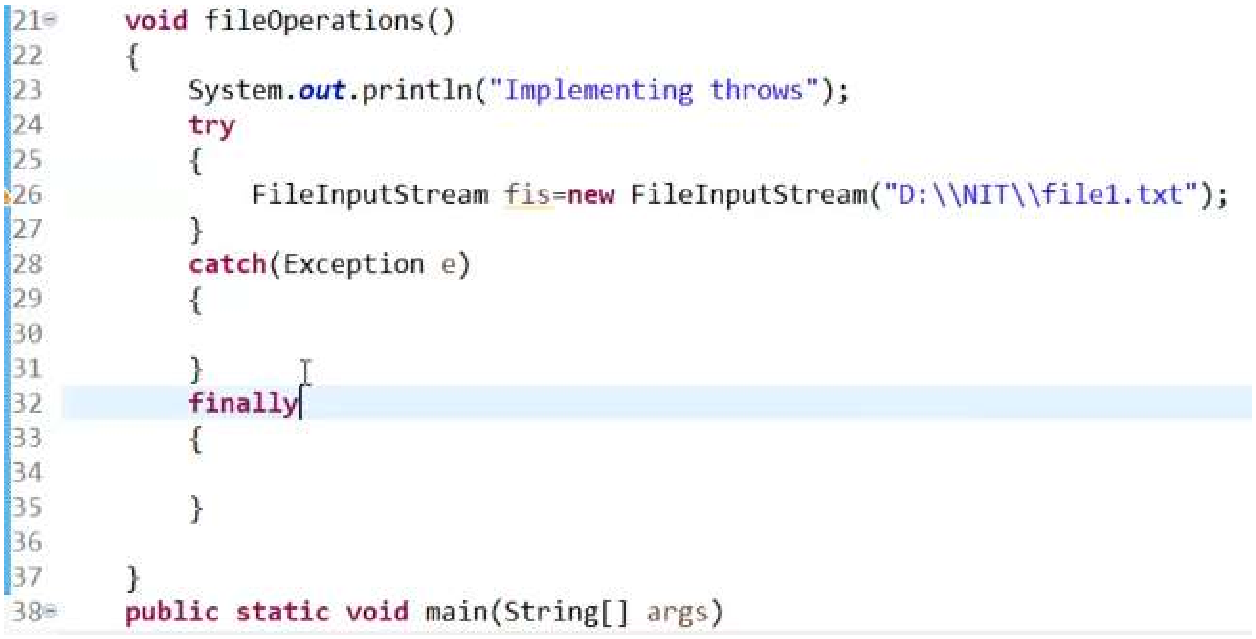
```
Exception in thread "main" java.lang.IllegalArgum
        at Training/com.pack1.ClassA.withDraw(Cla
        at Training/com.pack1.ClassA.main(ClassA.
```



```
    void fileOperations()
    {
        System.out.println("Implementing throws");
        FileInputStream fis=new FileInputStream("D:\\NIT\\file1.txt");
    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        aobj.withDraw(6000);
    }
}
```

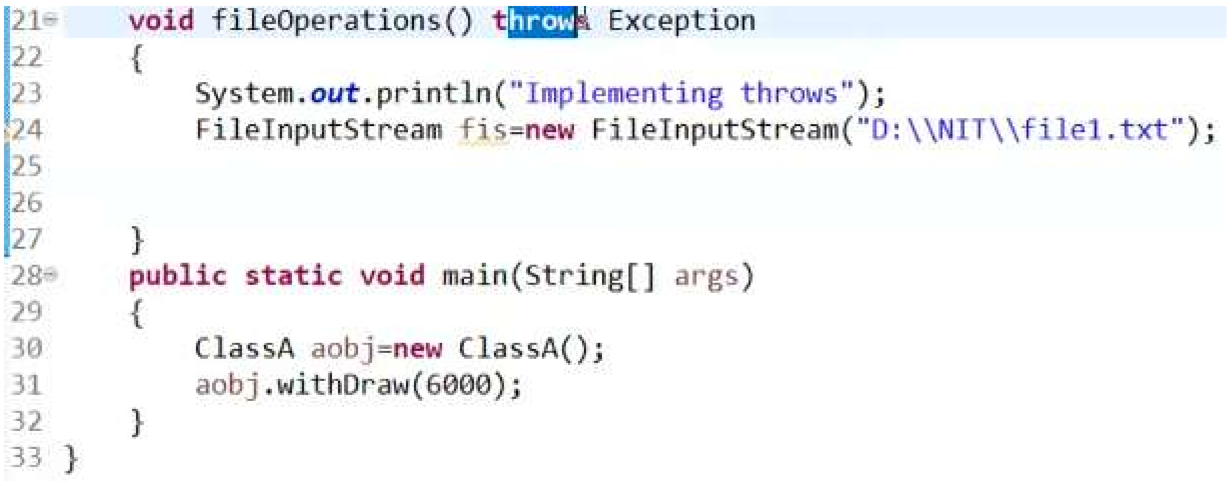
The above is checked exception we know the file is present, but compiler doesn’t know whether file is present or not

During the run time only knows if the file is presented or not

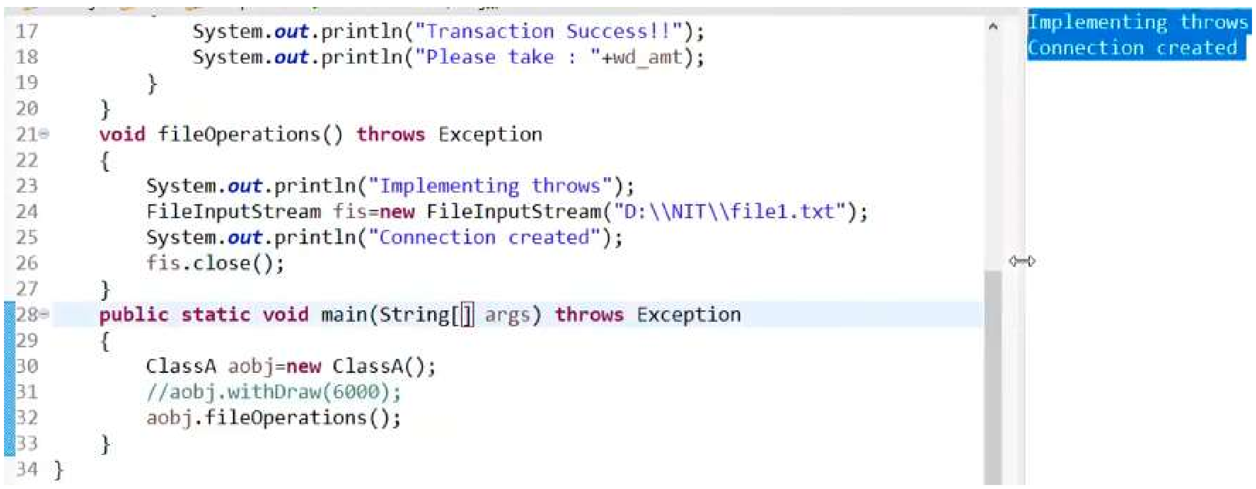


```java
    void fileOperations()
    {
        System.out.println("Implementing throws");
        try
        {
            FileInputStream fis=new FileInputStream("D:\\NIT\\file1.txt");
        }
        catch(Exception e)
        {

        }
        finally
        {

        }

    }
    public static void main(String[] args)
```

The error is resolved. I know the file is presented, then what is the used of writing the try catch and finally block.

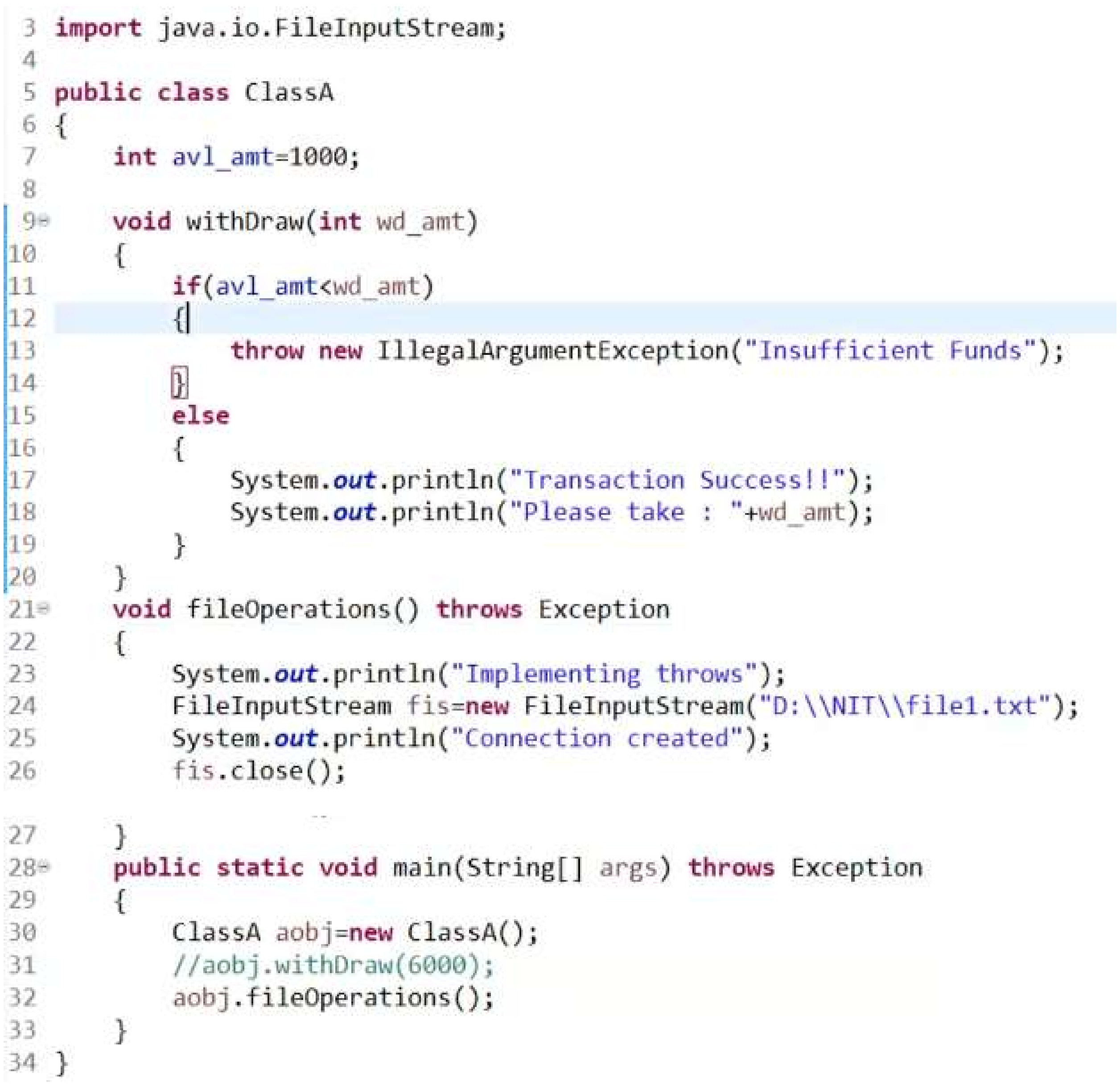


```java
    void fileOperations() throws Exception
    {
        System.out.println("Implementing throws");
        FileInputStream fis=new FileInputStream("D:\\NIT\\file1.txt");


    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        aobj.withDraw(6000);
    }
}
```



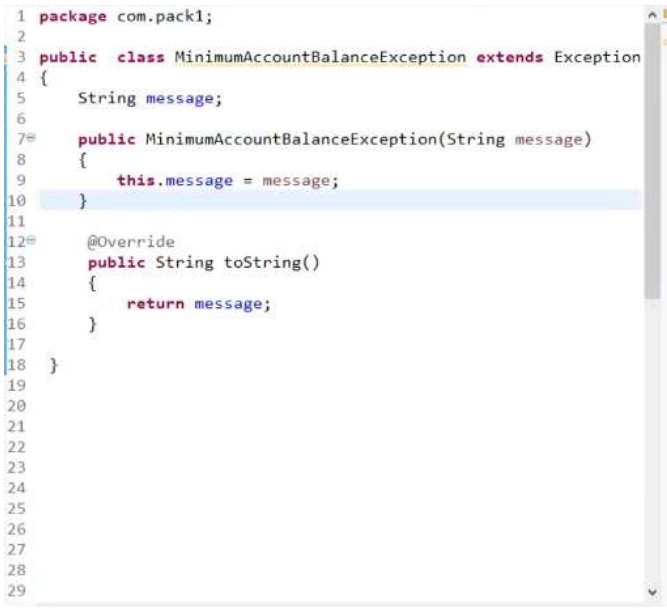
```
            System.out.println("Transaction Success!!");
            System.out.println("Please take : "+wd_amt);
        }
    }
    void fileOperations() throws Exception
    {
        System.out.println("Implementing throws");
        FileInputStream fis=new FileInputStream("D:\\NIT\\file1.txt");
        System.out.println("Connection created");
        fis.close();
    }
    public static void main(String[] args) throws Exception
    {
        ClassA aobj=new ClassA();
        //aobj.withDraw(6000);
        aobj.fileOperations();
    }
}
```

```
Implementing throws
Connection created
```

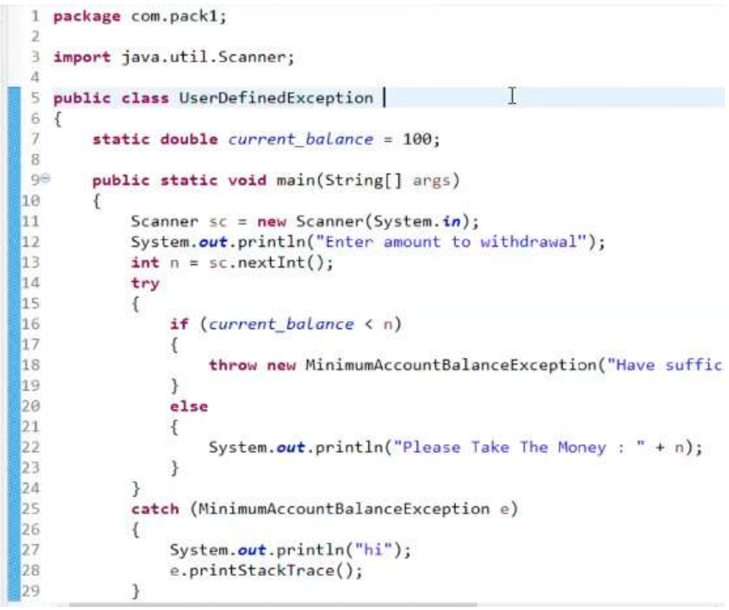


```java
import java.io.FileInputStream;

public class ClassA
{
    int avl_amt=1000;

    void withDraw(int wd_amt)
    {
        if(avl_amt<wd_amt)
        {
            throw new IllegalArgumentException("Insufficient Funds");
        }
        else
        {
            System.out.println("Transaction Success!!");
            System.out.println("Please take : "+wd_amt);
        }
    }
    void fileOperations() throws Exception
    {
        System.out.println("Implementing throws");
        FileInputStream fis=new FileInputStream("D:\\NIT\\file1.txt");
        System.out.println("Connection created");
        fis.close();

    }
    public static void main(String[] args) throws Exception
    {
        ClassA aobj=new ClassA();
        //aobj.withDraw(6000);
        aobj.fileOperations();
    }
}
```

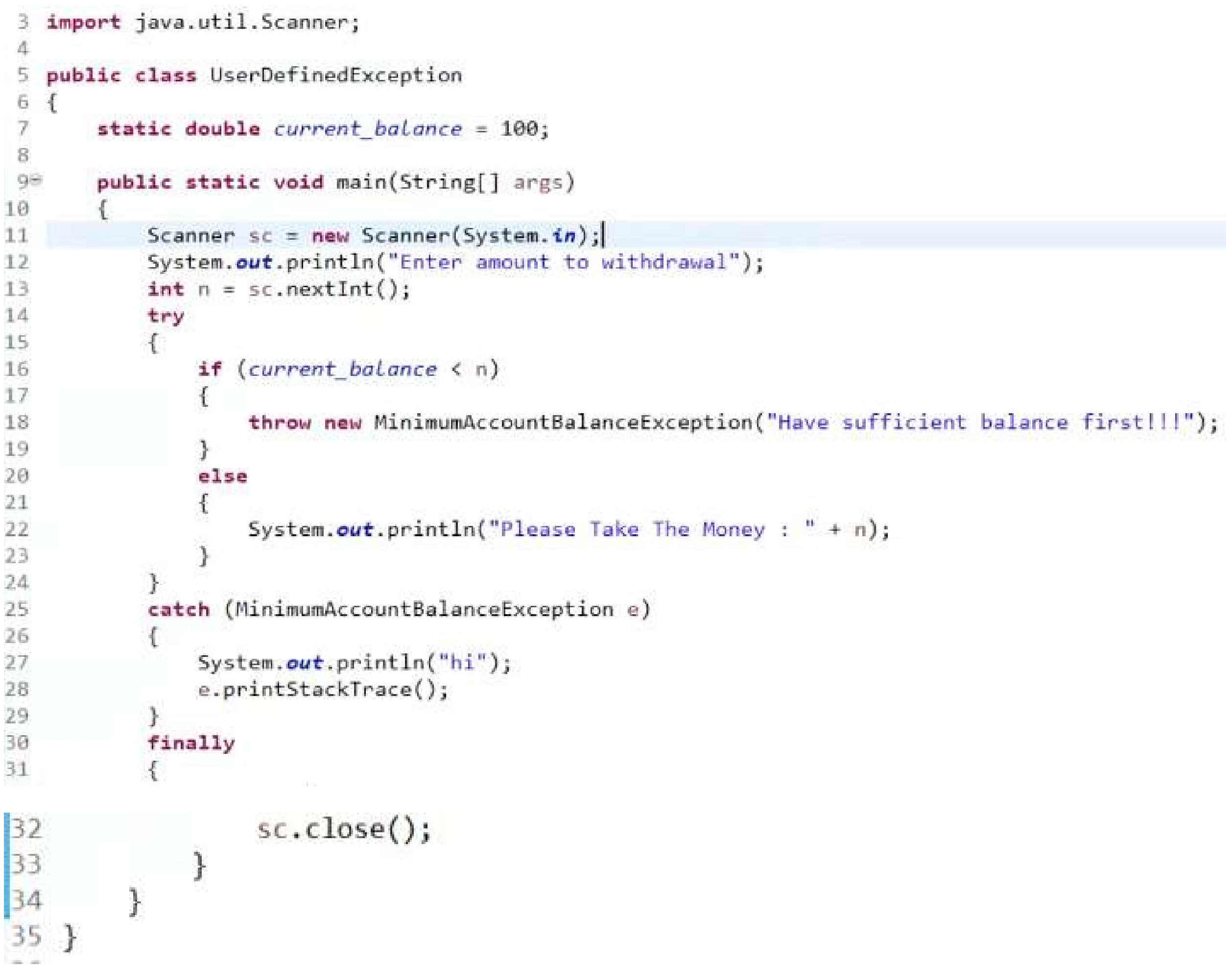
Creating user defined Exception Class



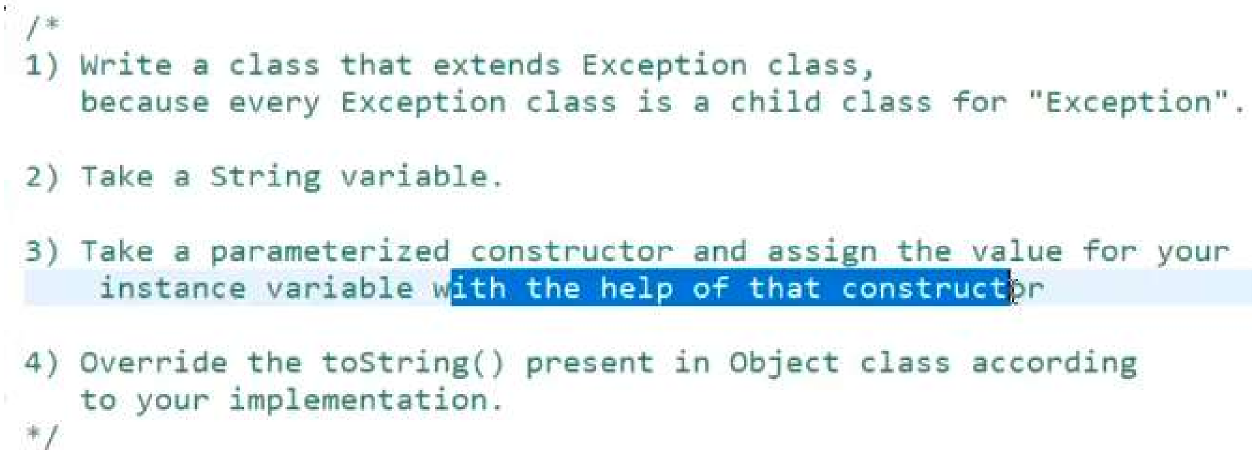
```java
package com.pack1;

public  class MinimumAccountBalanceException extends Exception
{
    String message;

    public MinimumAccountBalanceException(String message)
    {
        this.message = message;
    }

     @Override
     public String toString()
     {
         return message;
     }

 }
```



```java
package com.pack1;

import java.util.Scanner;

public class UserDefinedException
{
    static double current_balance = 100;

    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter amount to withdrawal");
        int n = sc.nextInt();
        try
        {
            if (current_balance < n)
            {
                throw new MinimumAccountBalanceException("Have suffic
            }
            else
            {
                System.out.println("Please Take The Money : " + n);
            }
        }
        catch (MinimumAccountBalanceException e)
        {
            System.out.println("hi");
            e.printStackTrace();
        }
```



```java
import java.util.Scanner;

public class UserDefinedException
{
    static double current_balance = 100;

    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter amount to withdrawal");
        int n = sc.nextInt();
        try
        {
            if (current_balance < n)
            {
                throw new MinimumAccountBalanceException("Have sufficient balance first!!!");
            }
            else
            {
                System.out.println("Please Take The Money : " + n);
            }
        }
        catch (MinimumAccountBalanceException e)
        {
            System.out.println("hi");
            e.printStackTrace();
        }
        finally
        {
            sc.close();
        }
    }
}
```



```
/*
1) Write a class that extends Exception class,
   because every Exception class is a child class for "Exception".

2) Take a String variable.

3) Take a parameterized constructor and assign the value for your
    instance variable with the help of that constructor

4) Override the toString() present in Object class according
   to your implementation.
*/
```