Try with multiple catch Blocks

The way of handling the exception is varied from exception to exception,hence for every exception type it is recommended to take a separate catch block. That is try with multiple catch blocks is possible and recommended to use.

Example 1

```
try{
  ...
  ...
  ...
}
catch(Exception e){
 default handler
}
```

This approach is not recommended because for any type of Exception we are using the same catch block.

Example 2

```
try{
  ....
  ....
  ....
}catch(FileNotFoundException fe){

}catch(ArithmeticException ae){

}catch(SQLException se){

}catch(Exception e){

}
```

- This approach is highly recommended because for any exception raise we are defining a separate catch block.
- If try with multiple catch blocks present then order of catch blocks is very important, it should be from child to parent.
- By mistake if we are taking from parent to child then we will get **"CompileTimeError" saying "exception XXXX has already been caught".**

Example 1

```
class Test{
 public static void main(String[] args){
  try{
   System.out.println(10/0);
  }catch(Exception e){
   e.printStackTrace();
  }catch(ArithmeticException ae){
   ae.printStackTrace();
  }
 }
}
```

CE: exception java.lang.ArithmeticException has already been caught

Example 2

```
class Test{
 public static void main(String[] args){
  try{
   System.out.println(10/0);
  }catch(ArithmeticException ae){
   ae.printStackTrace();
  }catch(Exception e){
   e.printStackTrace();
  }
 }
}
```

Output: Compile successfully

finally block

- It is not recommended to clean up code inside a try block because there is no guarantee for the execution of every statement inside a try block.
- It is not recommended to place clean up code inside the catch block becoz if there is no exception then the catch block won't be executed.
- we require some place to maintain clean up code which should be executed always irrespective of whether exceptions are raised or not raised and whether or not handled.
- Such type of best place is nothing but finally block.
- Hence the main objective of finally block is to maintain cleanup code.

Example

```
try{
 risky code
}catch( X e){
 handling code
}finally{
 cleanup code
}
```

The speciality of finally block is it will be executed always irrespective of whether the exception is raised or not raised and whether handled or not handled.

Case-1: If there is no Exception

```
class Test{
 public static void main(String... args){
  try{
   System.out.println("try block gets executed");
  }catch(ArithmeticException e){
   System.out.println("catch block gets executed");
  }finally{
   System.out.println("finally block gets executed");
  }
 }
}
```

Output:
try block gets executed
finally block gets executed

Case-2: If an Exception is raised,but the corresponding catch block matched

```
class Test{
 public static void main(String... args){
  try{
   System.out.println("try block gets executed");
   System.out.println(10/0);
  }catch(ArithmeticException e){
   System.out.println("catch block gets executed");
  }finally{
   System.out.println("finally block gets executed");
  }
 }
}
```

Output:
try block gets executed
catch block gets executed
finally block gets executed

Case-3: If an Exception is raised,but the corresponding catch block not matched

```
class Test{
 public static void main(String... args){
  try{
   System.out.println("try block gets executed");
   System.out.println(10/0);
  }catch(NullPointerException e){
   System.out.println("catch block gets executed");
  }finally{
   System.out.println("finally block gets executed");
  }
 }
}
```

Output:
Try block gets executed
finally block gets executed
Exception in thread "main" java.lang.ArithemeticException :/by Zero atTest.main(Test.java:8)

return vs finally

- Even though the return statement present in try or catch blocks first finally will be executed and after that only return statement will be considered
- finally block dominates return statement.

Example

```
class Test{

 public static void main(String... args){
  try{
   System.out.println("try block executed");
   return;
  }catch(ArithmeticException e){
   System.out.println("catch block executed");
  }finally{
   System.out.println("finally block executed");
  }
 }
}
```

Output:
try block executed
finally block executed

Example
If the return statement present try,catch and finally blocks then finally block return statement will be considered.

```
class Test{
 public static void main(String... args){
  System.out.println(m1());
 }
 public static int m1(){
  try{
   System.out.println(10/0);
   return 777;
  }catch(ArithmeticException e){
   return 888;
  }finally{
   return 999;
  }
        }
}
```

inally vs System.exit(0)

- There is only one situation where the finally block won't be executed whenever we are using System.exit(0) method.
- Whenever we are using System.exit(0) then the JVM itself will be shutdown, in this case the finally block won't be executed.
  - ie,.. System.exit(0) dominates finally block

System.exit(0)

- This argument acts as status code, Instead of Zero, we can't take any integer value
- Zero means normal termination, non zero means abnormal termination
- This status code internally used by JVM,whether it is zero or non-zero there is no change in the result and effect is the same w.r.t program

Difference b/w final,finally and finalize

final

- final is the modifier applicable for classes, methods and variables
- If a class is declared as the final then child class creation is not possible.
- If a method is declared as the final then overriding of that method is not possible.
- If a variable is declared as the final then reassignment is not possible.

finally

- It is a final block associated with try-catch to maintain clean up code, which should be executed always irrespective of whether exceptions are raised or not raised and whether handled or not handled.

finalize

- It is a method, always invoked by Garbage Collector just before destroying an object to perform cleanup activities.

Note

- finally block meant for cleanup activities related to try block whereas finalize() method for cleanup activities related to object.
- To maintain cleanup code finally block is recommended over finalize() method because we can't expect exact behaviour of GC.

Handling vs Ducking an Exception

- It is highly recommended to handle exceptions
- In our program the code which may rise exception is called "risky code"
- We have to place our risky code inside the try block and corresponding handling code inside the catch block.

Syntax

```
try{
      ...
      ... risky code
      ...
 }catch(XXXX e){
     ...
     ... handling code
     ...
}
```