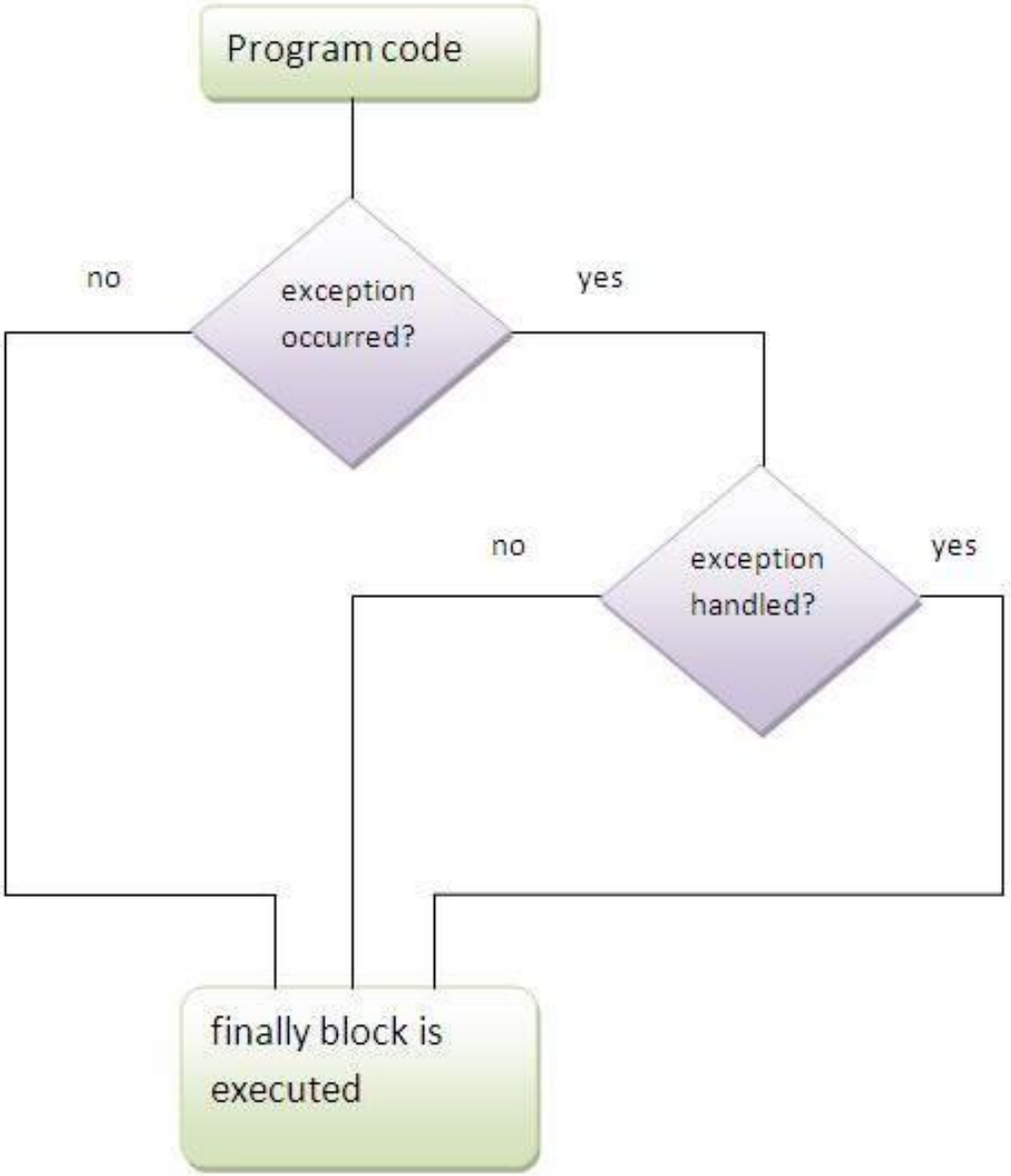
Programming Language II
CSE-215

Prof. Dr. Mohammad Abu Yousuf
yousuf@juniv.edu

Java finally block

- **Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc.
- Java finally block is always executed whether exception is handled or not.
- Java finally block follows try or catch block.
- Note: **If you don't handle exception, before terminating the program, JVM executes finally block(if any).**

Java finally block



Usage of Java finally

Case 1

- Java finally example where **exception doesn't occur**.

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
   System.out.println("rest of the code...");
  }
}
```

Output:5
finally block is always executed
rest of the code...

Usage of Java finally

Case 2

- Java finally example where **exception occurs and not handled**.

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Output: finally block is always executed
Exception in thread main java.lang.ArithmeticException:/ by zero

Usage of Java finally

Case 3

- Java finally example where **exception occurs and handled**.

```
public class TestFinallyBlock2{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}

  System.out.println("rest of the code...");
  }
}
```

Output: Exception in thread main java.lang.ArithmeticException:/ by zero
finally block is always executed
rest of the code...

Usage of Java finally

- **Rule:** For each try block there can be zero or more catch blocks, but only one finally block.

- **Note:** The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

Difference between final, finally and finalize

- Differences between final, finally and finalize are given below:

No.	final	finally	finalize
1)	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

Difference between final, finally and finalize

- Java final example

```
class FinalExample{
public static void main(String[] args){
final int x=100;
x=200;//Compile Time Error
}
}
```

Difference between final, finally and finalize

- Java finally example

```
class FinallyExample{
public static void main(String[] args){
try{
int x=300;
}catch(Exception e){System.out.println(e);}
finally{System.out.println("finally block is executed");}
}
}
```

Difference between final, finally and finalize

- Java finalize example

```
class FinalizeExample{
public void finalize(){System.out.println("finalize called");}
public static void main(String[] args){
FinalizeExample f1=new FinalizeExample();
FinalizeExample f2=new FinalizeExample();
f1=null;
f2=null;
System.gc();
}
}
```

Exception Handling with Method Overriding in Java

- There are many rules if we talk about method overriding with exception handling. The Rules are as follows:
- **If the superclass method does not declare an exception:**
  - If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
- **If the superclass method declares an exception:**
  - If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

Exception Handling with Method Overriding in Java

- **If the superclass method does not declare an exception**

  1) Rule: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception.

```
import java.io.*;
class Parent{
  void msg(){System.out.println("parent");}
}
class TestExceptionChild extends Parent{
  void msg()throws IOException{
    System.out.println("TestExceptionChild");
  }
  public static void main(String args[]){
   Parent p=new TestExceptionChild();
   p.msg();
  }
}
```

Output:
Compile Time Error

Exception Handling with Method Overriding in Java

- **If the superclass method does not declare an exception**

  2) Rule: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.

```
import java.io.*;
class Parent{
  void msg(){System.out.println("parent");}
}
class TestExceptionChild1 extends Parent{
  void msg()throws ArithmeticException{
    System.out.println("child");
  }
  public static void main(String args[]){
   Parent p=new TestExceptionChild1();
   p.msg();
  }
}
```

Output:
child

Exception Handling with Method Overriding in Java

- **If the superclass method declares an exception**

  1) Rule: If the superclass method declares an exception, subclass overridden method can declare same exception, subclass exception or no exception but cannot declare parent exception.

```
import java.io.*;
class Parent{
  void msg()throws ArithmeticException{System.out.println("parent");}
}
class TestExceptionChild2 extends Parent{
  void msg()throws Exception{System.out.println("child");}
  public static void main(String args[]){
   Parent p=new TestExceptionChild2();
   try{
   p.msg();
   }catch(Exception e){}
  }
}
```

Example in case subclass overridden method declares parent exception

Output: Compile Time Error

Exception Handling with Method Overriding in Java

- **If the superclass method declares an exception**

```
import java.io.*;
class Parent{
  void msg()throws Exception{System.out.println("parent")
;}
}
class TestExceptionChild3 extends Parent{
  void msg()throws Exception{System.out.println("child");}
  public static void main(String args[]){
   Parent p=new TestExceptionChild3();
   try{
   p.msg();
   }catch(Exception e){}
  }
}
```

Example in case subclass overridden method declares same exception

Output:
child

Exception Handling with Method Overriding in Java

- **If the superclass method declares an exception**

```
import java.io.*;
class Parent{
  void msg()throws Exception{System.out.println("parent");}
}

class TestExceptionChild4 extends Parent{
  void msg()throws ArithmeticException{System.out.println("child");}

  public static void main(String args[]){
   Parent p=new TestExceptionChild4();
   try{
   p.msg();
   }catch(Exception e){}
  }
}
```

Example in case subclass overridden method declares subclass exception

Output:
child

Exception Handling with Method Overriding in Java

- **If the superclass method declares an exception**

```
import java.io.*;
class Parent{
  void msg()throws Exception{System.out.println("parent");}
}
class TestExceptionChild5 extends Parent{
  void msg(){System.out.println("child");}
  public static void main(String args[]){
   Parent p=new TestExceptionChild5();
   try{
   p.msg();
   }catch(Exception e){}
  }
}
```

Example in case subclass overridden method declares no exception

Output:
child

Java Custom Exception

- If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

- By the help of custom exception, you can have your own exception and message.

How to create a custom exception

Writing your own exception class

- Create a new class whose name should end with Exception like ClassNameException. This is a convention to differentiate an exception class from regular ones.

- Make the class extends one of the exceptions which are subtypes of the java.lang.Exception class. Generally, a custom exception class always extends directly from the Exception class.

- Create a constructor with a String parameter which is the detail message of the exception. In this constructor, simply call the super constructor and pass the message.

```java
// A Class that represents use-defined expception
class MyException extends Exception
{
    public MyException(String s)
    {  // Call constructor of parent Exception
       super(s);   }
}
// A Class that uses above MyException
public class Main
{  // Driver Program
    public static void main(String args[])
    {  try
       {  // Throw an object of user defined exception
          throw new MyException("GeeksGeeks");
       }
       catch (MyException ex)
       {  System.out.println("Caught");
          // Print the message from MyException object
          System.out.println(ex.getMessage());
       }
    }
}
```

example of java custom exception

Output:

```
Caught
GeeksGeeks
```

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 }
}
```

example of java custom exception

```
class TestCustomException1{
   static void validate(int age)throws InvalidAgeException{
     if(age<18)
      throw new InvalidAgeException("not valid");
     else
      System.out.println("welcome to vote");
   }
   public static void main(String args[]){
      try{
      validate(13);
      }catch(Exception m){System.out.println("Exception occured: "+m);}

      System.out.println("rest of the code...");
  }
}
```

Output of previous program:

Exception occured: InvalidAgeException:not valid
rest of the code...

What is wrong with following code?

```
public static void start() throws IOException, RuntimeException{
   throw new RuntimeException("Not able to Start");
 }
 public static void main(String args[]) {
   try {
        start();
   } catch (Exception ex) {
         ex.printStackTrace();
   } catch (RuntimeException re) {
         re.printStackTrace();
   }
 }
```

Answer:

- This code will throw compiler error on line where RuntimeException variable “re” is written on catch block. since Exception is super class of RuntimeException, all RuntimeException thrown by start() method will be captured by first catch block and code will never reach second catch block and that's the reason compiler will flag error as *“exception java.lang.RuntimeExceptionhas already been caught"*.

What is the problem with below program?

```
package com.journaldev.exceptions;
import java.io.IOException;
public class TestException4 {
    public void start() throws IOException{
    }
    public void foo() throws NullPointerException{
    }
}

class TestException5 extends TestException4{
    public void start() throws Exception{
    }
    public void foo() throws RuntimeException{ }
    }
```

What is the problem with below program?

- The above program won't compile because start() method signature is not same in subclass. To fix this issue, we can either change the method signature in subclass to be exact same as superclass or we can remove throws clause from subclass method as shown below.

```
public void start(){
    }
```

Thank You