Lecture Five

Exception Handling

C++ & Java

© **Dr. Mohammad Mahfuzul Islam, PEng**
Professor, Dept. of CSE, BUET

Exception Handling

- An **Exception** is a **run-time error**. Exception handling is a systematic approach for managing code errors.

C++ Keywords for Exception Handling:

- ✓ **try**
- ✓ **catch**
- ✓ **throw**

Java Keywords for Exception Handling:

- ✓ **try**
- ✓ **catch**
- ✓ **throw**
- ✓ **throws**
- ✓ **finally**

- If an **Exception** is not caught, the **run-time system** aborts the program (i.e., crash). In C++, the standard library function **terminate()** is invoked which call **abort()** to **stop the program**.

Exception Handling

C++ Code (ExceptionError.cpp)

```cpp
#include <iostream>
using namespace std;

int divide(){
   int d = 0;
   int a = 10 /d;
   return a;
}

int main(){
   cout << divide() << endl;
   return 0;
}
```

OUTPUT:

```
zsh: floating point exception
./"ExceptionError"
```

Java Code

```java
public class Main {
    static int divide(){
        int d = 0;
        int a = 10 / d;
        return a;
    }
    public static void main(String[] args) {
       System.out.println(Main.divide());
    }
}
```

OUTPUT:

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
       at Main.divide(Main.java:4)
       at Main.main(Main.java:8)
```

Exception Handling

C++ Code

```
#include <iostream>
#include <exception>
using namespace std;

int divide(){
   int d = 0;
   if (d == 0){
       throw runtime_error("Divide by zero");
    }
   int a = 10 /d;
   return a;
}

int main(){
   try{
      cout << divide() << endl;
   } catch (exception &e){
      cout << "Exception caught: " << e.what() << endl;
   }
   cout << “After catch” << endl;
   return 0;
}
```

OUTPUT:

```
Exception caught: Divide by zero
After catch
```

Java Code

```
public class Main {

  static int divide(){
     int d = 0;
     int a = 10 / d;
     return a;
  }

  public static void main(String[] args) {
     try {
        System.out.println(Main.divide());
     } catch (RuntimeException e) {
        System.out.println("Caught Exception: "+ e);
     }
     System.out.println("After catch");
  }
}
```

OUTPUT:

```
Caught Exception:
java.lang.ArithmeticException: / by zero
After catch
```

Use of try, catch and throw in C++

```
#include<iostream>
using namespace std;

enum ErrorType{InvalidInput,DivideByZero,OutOfRange};

void Xtest(int test){
    if (!test) throw "zero";
    if(test<0) throw InvalidInput;
    else throw test;
}

int main(){
   int a = 10;
   try{
     if(a==0) throw DivideByZero;
     else if(a>10) throw OutOfRange;
     else throw a;

     Xtest(0);
     Xtest(-5);
     Xtest(2);
     Xtest(20);
   }
```

C++ Code

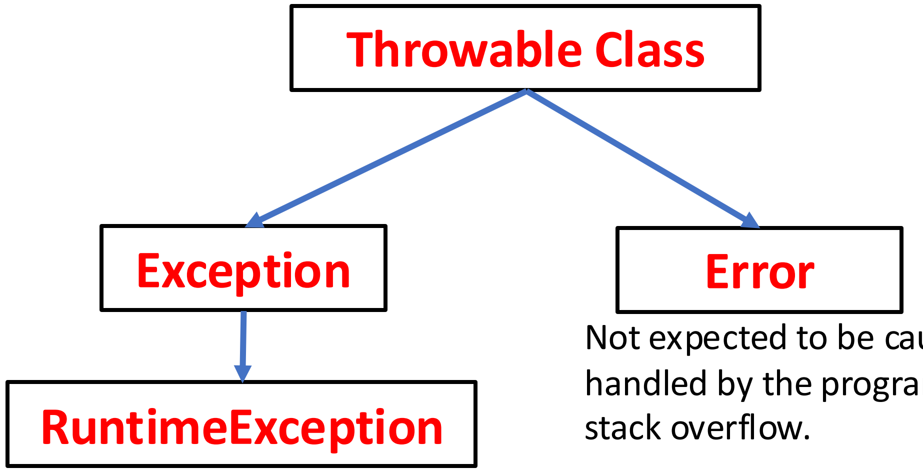
```
   catch(ErrorType e){
       if(e==InvalidInput){
           cout << "Invalid Input" << endl;
       }
       else if(e==DivideByZero){
           cout << "Divide By Zero" << endl;
       }
       else if(e==OutOfRange){
           cout << "Out of Range" << endl;
       }
   }

   catch(int i){
       cout << "Caught an integer: " << i << endl;
   }

   catch(...){
       cout << "Caught a String. " << endl;
   }

   return 0;
}
```

OUTPUT:

```
Caught an integer: 10
```

Use of try, catch and throw in Java

- Java **cannot throw primitive type** or **non-throwable** class such as String or Object.
- **Throwable** class is declared under **java.lang** package.
- There is **no reason** to extend **Throwable class**; rather **Exception** or an **existing subclass of Exception** can be extended.



Not expected to be caught and handled by the program. Example, stack overflow.

```java
class MyException extends Exception {
   public MyException(String str){
      super(str);
   }
}

public class Main{
   public static void main(String args[]){
     try{
        throw new MyException(”User-defined exception.");
     }
     catch (MyException e){
        System.out.println("Caught the exception");
        System.out.println(e.getMessage());
     }
   }
}
```

OUTPUT:

```
Caught the exception
User-defined exception
```

Function throw Exception in C++

- ✓ Compile-time check only;
- ✓ The compiler can use this information to enable certain optimizations.

```
void Xtest(int test){
    if (!test) throw "zero";
    if(test<0) throw InvalidInput;
    else throw test;
}
```

Both works

```
void Xtest(int test) throw(int, char, double, ErrorType){
    if (!test) throw "zero";
    if(test<0) throw InvalidInput;
    else throw test;
}
```

Function throws Exception in Java

```
class ThrowsDemo {

   static void throwOne() {
      System.out.println("Inside throwOne.");
      throw new IllegalAccessException("demo");
   }

   public static void main(String[] args) {
      try {
         throwOne();
      } catch(IllegalAccessException e){
         System.out.println("Caught: "+e);
      }
   }
}
```

OUTPUT:

```
java: unreported exception
java.lang.IllegalAccessException; must be caught or
declared to be thrown
```

```
class ThrowsDemo {

   static void throwOne() throws IllegalAccessException {
      System.out.println("Inside throwOne.");
      throw new IllegalAccessException("demo");
   }

   public static void main(String[] args) {
      try {
         throwOne();
      } catch(IllegalAccessException e){
         System.out.println("Caught: "+e);
      }
   }
}
```

OUTPUT:

```
Inside throwOne.
Caught: java.lang.IllegalAccessException: demo
```

Re-throw Exception

C++ Code

- ✓ An exception can be rethrown from within a catch block.
- ✓ When an exception is rethrown, it will not be recaught by the same catch statement.

```cpp
#include <iostream>
using namespace std;

void Xtest(){
    try{
        throw "String";
    }
    catch(const char* s){
        cout << "Caught a string: " << s << endl;
        throw;   // throw s; -> ok & the same
    }
}

int main(){
    try{
        Xtest();
    }
    catch(const char* s){
        cout << "Caught a string in main: " << s << endl;
    }
    return 0;
}
```

OUTPUT:

```
Caught a string: String
Caught a string in main: String
```

Re-throw Exception

```
public class Main {

    static void XTest() throws RuntimeException{
        try{
            throw new RuntimeException("Runtime Exception.");
        }catch (RuntimeException e){
            System.out.println("Caught in XTest: "+e);
            throw e;  // only throw is not work
        }
    }

    public static void main(String[] args) {
        try{
            XTest();
        } catch (RuntimeException e){
            System.out.println("Caught in main: "+e);
        }
    }
}
```

OUTPUT:

```
Caught in XTest: java.lang.RuntimeException: Runtime Exception.
Caught in main: java.lang.RuntimeException: Runtime Exception.
```

Use of finally keyword

- ✓ **finally** block is executed whether or not an exception is thrown;
- ✓ **finally** block is executed after try/catch block and before returning from function.

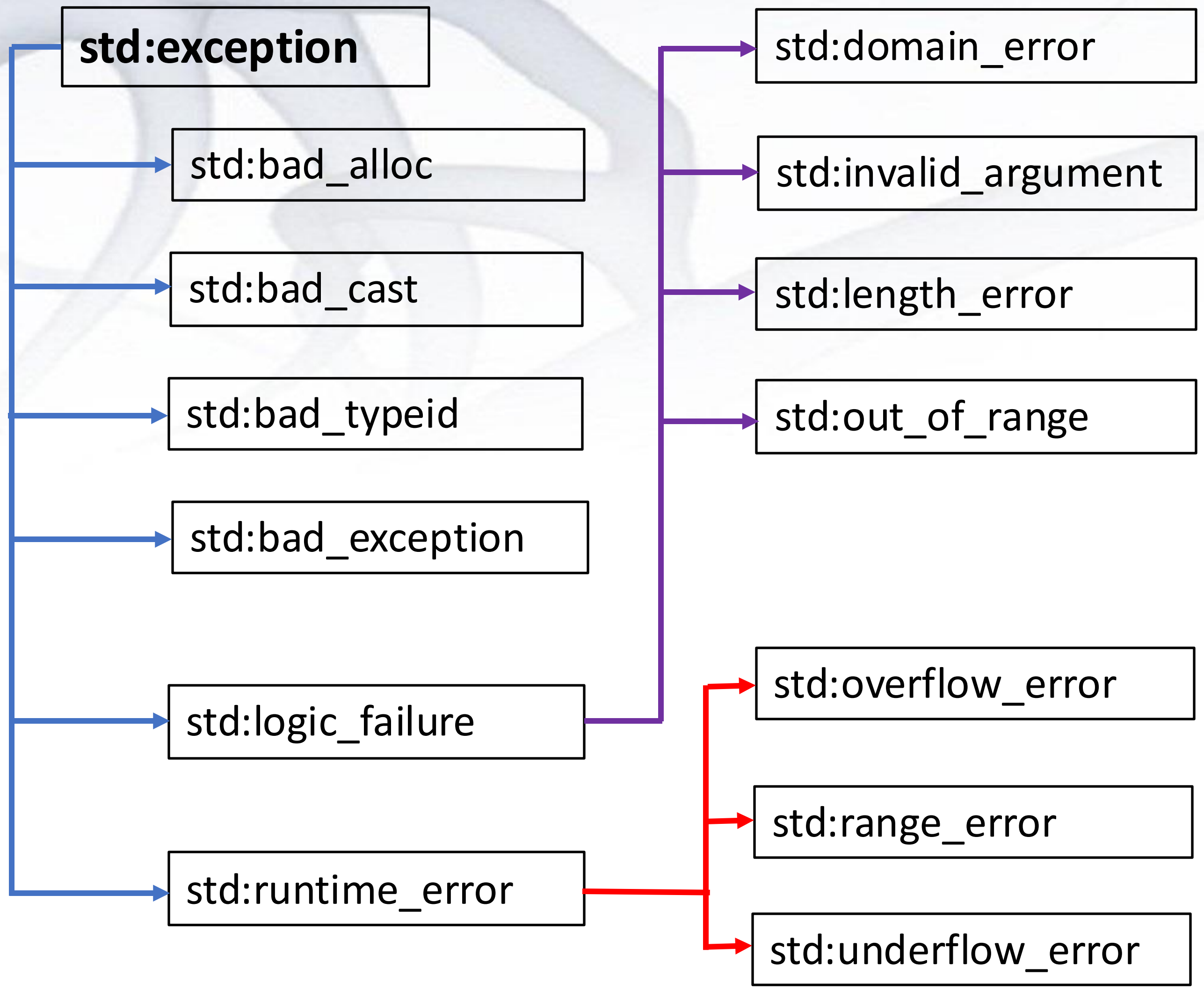
```
class FinallyDemo {

  static void procA() {
    try {
      System.out.println("inside procA");
      throw new RuntimeException("demo");
    } finally {
      System.out.println("procA's finally");
    }
  }

  static void procB() {
    try {
      System.out.println("inside procB");
      return;
    } finally {
      System.out.println("procB's finally");
    }
  }
```

```
  static void procC() {
      try {
        System.out.println("inside procC");
      } finally {
        System.out.println("procC's finally");
      }
    }

  public static void main(String[] args) {
    try {
      procA();
    } catch (Exception e) {
      System.out.println("Exception caught");
    }
    procB();
    procC();
  }
}
```

OUTPUT:

```
inside procA
procA's finally
Exception caught
inside procB
procB's finally
inside procC
procC's finally
```

exception class in C++



- **new** operator throw an **bad_alloc** exception if an allocation request is fails.

```
double *p = new double(100);
```

- To have access to this exception, **<new>** header must be included in the program.
- In modern C++, the following form returns **NULL** instead of **throwing** an exception.

```
p_var = new(nothrow) type;
```

Java’s Built-in Exceptions

Subclasses of RuntimeException

ArithmeticException
ArrayIndexOutOfBoundsException
ArrayStoreException
ClassCastException
EnumConstantNotPresentException
IllegalArgumentException
IllegalCallerException
IllegalMonitorStateException
IllegalStateException
IllegalThreadStateException
IndexOutOfBoundsException
LayerInstantiationException
NegativeArraySizeException
NullPointerException
NumberFormatException
SecurityException
StringIndexOutOfBoundsException
TypeNotPresentException
UnsupportedOperationException

Checked Exception

ClassNotFoundException
CloneNotSupportedException
IllegalAccessException
InstantiationException
InterruptedException
NoSuchFieldException
NoSuchMethodException
ReflectiveOperationException

These are unchecked exceptions, because the compiler does not check to see if a method handles or throws these exceptions

Methods defined by Throwable

final void addSuppressed(Throwable *exc*)
Throwable fillInStackTrace( )
Throwable getCause( )
String getLocalizedMessage( )
String getMessage( )
StackTraceElement[ ] getStackTrace( )
final Throwable[ ] getSuppressed( )
Throwable initCause(Throwable *causeExc*)
void printStackTrace( )
void printStackTrace(PrintStream *stream*)
void printStackTrace(PrintWriter *stream*)
void setStackTrace(StackTraceElement[ ] *elements*)
String toString( )