Middlesex University London

Lecture 13

Exception handling

TODAY:

- Exception handling
- Try-catch-finally statements
- System exceptions
- Runtime exceptions
- Throws
- Throw

EXCEPTION HANDLING

- An exception is an event that disrupts the normal flow of the program
- Exception handling is a mechanism to handle runtime error such as class not found, IO, SQL, Remote etc..

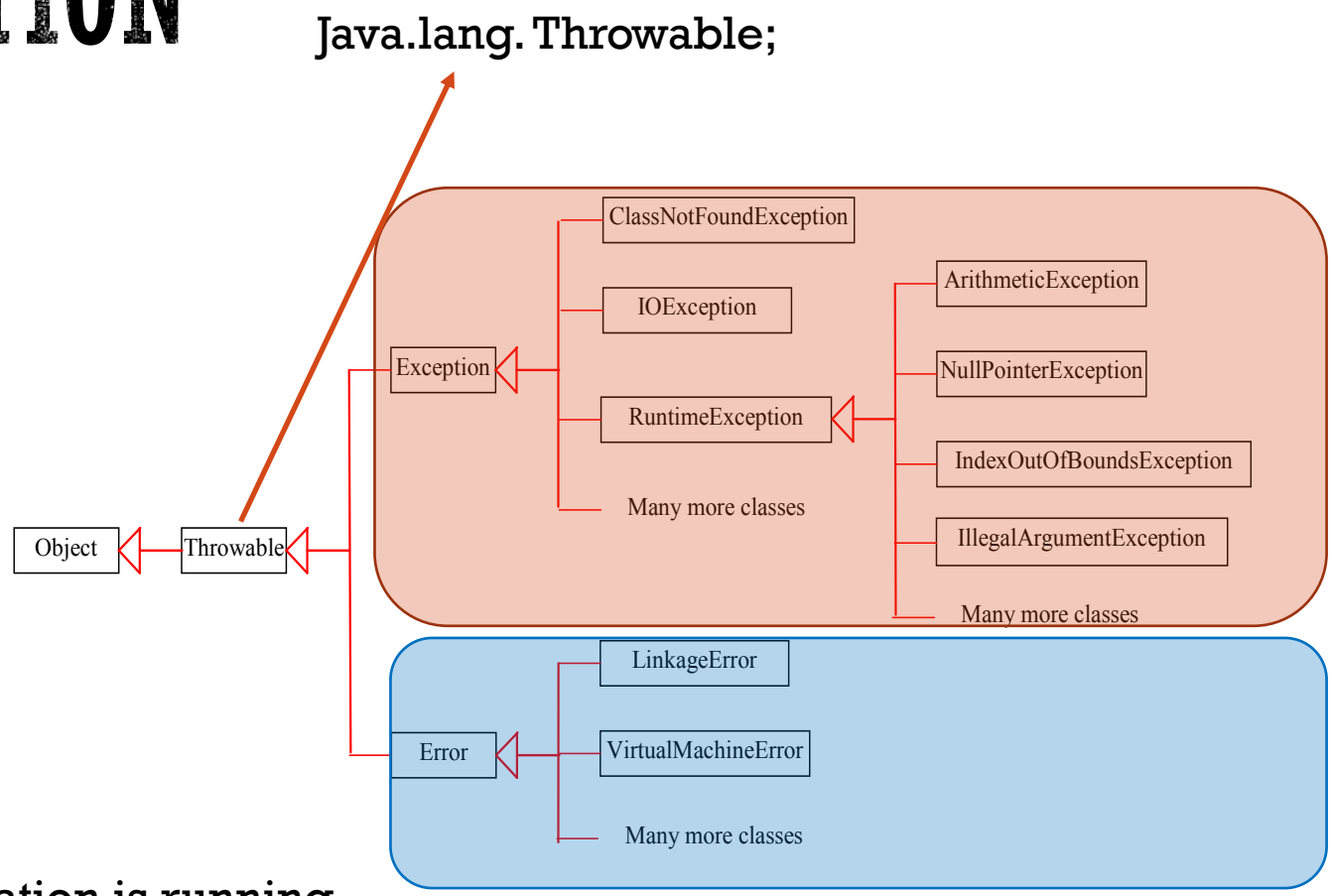
ERROR VS EXCEPTION

Exceptions:

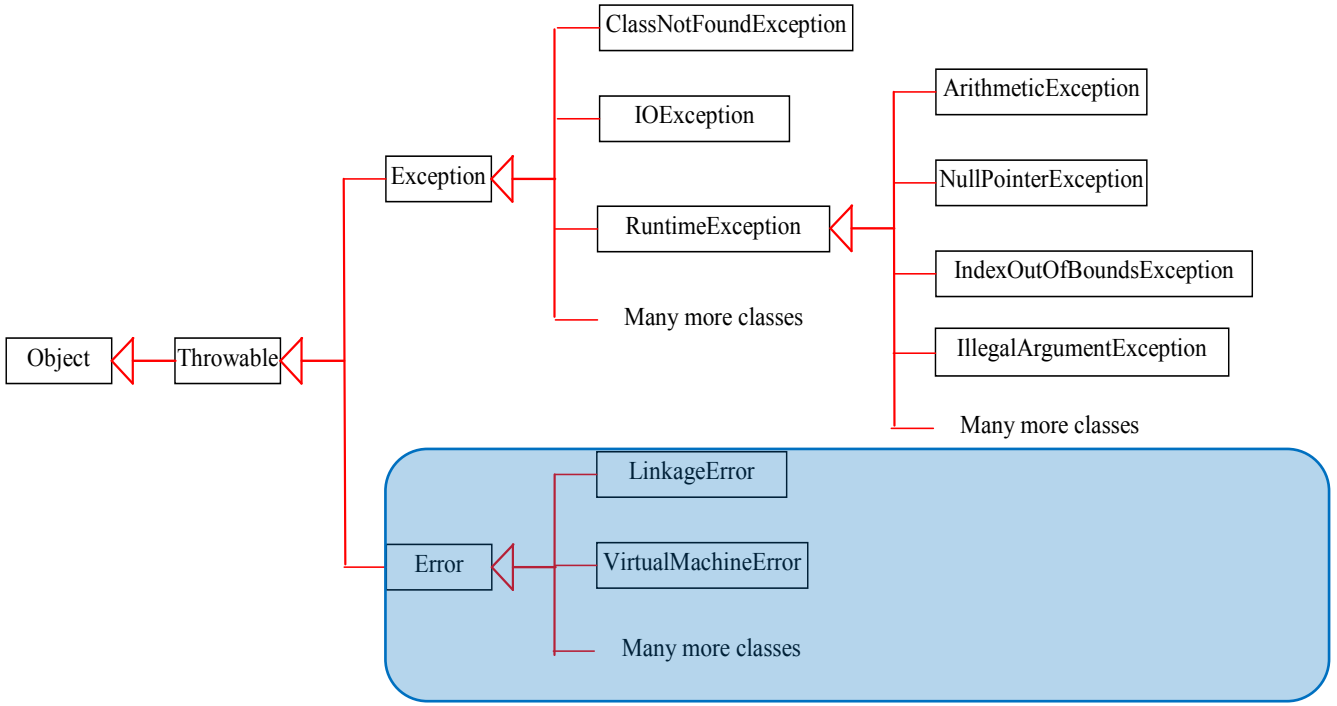
- Possible to recover from Exceptions
- It can be of checking or unchecked
- Can happen at Compile Time & Runtime
- Cause by application

Errors:

- It is imposable to recover from Error
- Errors are of Unchecked Type
- Happen at Run-time
- Cause by the environment on which application is running



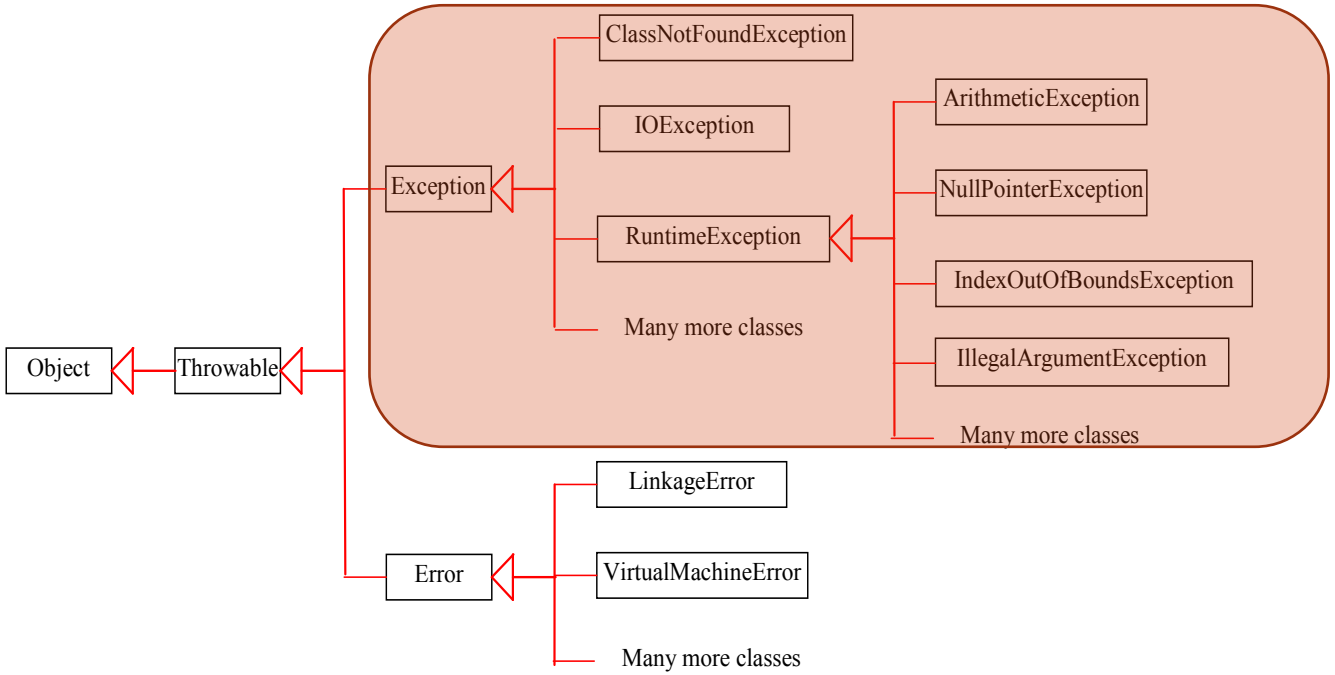
SYSTEM ERRORS



System errors are thrown by JVM and represented in the Error class. The Error class describes internal system errors. Such errors rarely occur. If one does, there is little you can do beyond notifying the user and trying to terminate the program gracefully.

EXCEPTIONS

Exception describes errors caused by your program and external circumstances. These errors can be caught and handled by your program.



TYPES OF EXCEPTIONS

Exception that happened at compile time (such is IO exception)

- An exception that it is check by compiler at compiler time
- This exceptions cannot be ignored, those should be handle by a program

Run time exceptions (such as divide by zero, bad casting, array out of bound, null pointer exceptions etc.)

- Occurs at a time of execution
- There are ignored at time of compilation

GENERAL STRUCTURE

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

    try{
        //program that may rase an exception

    }catch(Exception e){
        // rest of the program
    }
}
```

EXAMPLE:

Problem:

```
public static void main(String[] args) {
        int num1 = 3, num2 =0;
        int result = num1/num2;
        System.out.println(""+result);
}
```

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at week15_lecture.Exception.main(Exception.java:13)
Java Result: 1
```

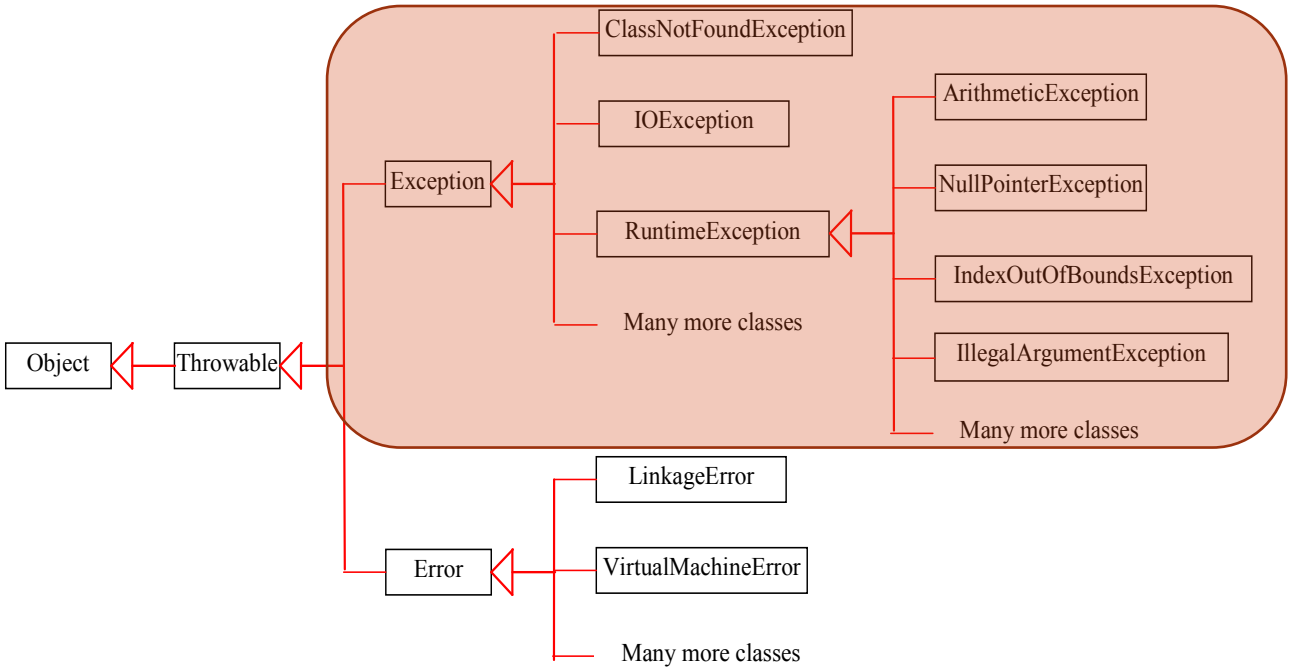
Solution:

```
public static void main(String[] args) {
    try{
        int num1 = 3, num2 =0;
        int result = num1/num2;
        System.out.println(""+result);
    }catch(ArithmeticException a){
        System.out.println("Cant devide by zero");
    }
```

```
run:
Cant devide by zero
BUILD SUCCESSFUL (total time: 0 seconds)
```

BUILD IN EXCEPTIONS

Are the one that are available by Java library



EXAMPLE:

Problem:

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

    int num = Integer.parseInt("Hello");
}
```

```
run:
Exception in thread "main" java.lang.NumberFormatException: For input string: "Hello"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.parseInt(Integer.java:615)
        at week15_lecture.Exception.main(Exception.java:14)
Java Result: 1
```

Solution:

```
    try{
        int num = Integer.parseInt("Hello");
        System.out.println(num);
    }catch(NumberFormatException e){
        System.out.println("Number format problem");
    }

}
```

```
Number format
BUILD SUCCESSFUL (total time: 0 seconds)
```

EXAMPLE:

Problem:

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

    int numbers [] =  new int[5];
    System.out.println(numbers[6]);

}
```

```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
        at week15_lecture.Exception.main(Exception.java:15)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Solution:

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

    try{
        int numbers [] =  new int[5];
        System.out.println(numbers[6]);
    }catch(ArrayIndexOutOfBoundsException e){
        System.out.println("the space does not exist");
    }
```

```
run:
the space does not exist
BUILD SUCCESSFUL (total time: 0 seconds)
```

METHODS IN EXCEPTION HANDLING

try

catch

finally

throw

throws

TRY

Used for a code including exception

```
try{
        // code that throw exception
} catch(Exception_class_name e){
        }
```

CATCH

```
try{
        // code that throw exception
} catch(Exception_class_name e){
        // catch block
} catch(Exception_class_name e){
        // catch block
} catch(Exception_class_name e){
}
```

FINALLY

```
try{
        // code that throw exception
} catch(Exception_class_name e){
        // catch block
} catch(Exception_class_name e){
        // catch block
} catch(Exception_class_name e){
} finally{
        // always is executed and it allows flow of the program

}
```

- It is a key word used to explicitly throw an exception
- Checked exception cannot be propagated using throw only
- Used with in a method

THROW

```
public static void method_name(){
throw new Exception_class_name (“statment”);
}
```

Example:

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

    int int1 =3, int2=0;
try{
    quotient (int1, int2);
}   catch(ArithmeticException ae){
    System.out.println("Exception: an integer cannot be devided by zero")
}

}

public static int quotient(int num1, int num2){
    if (num2 == 0){
      throw new ArithmeticException("you cannot devide by zero");
    }
    return num1/num2;

}
```

- Used to declare an exception
- It is used within method signature
- Can declare multiple exception

THROWS

```
public void method_name()throws Exception_class_name {

}
```

Example:

```
*/
public class Quotient {
    public static void main(String[] args)throws ArithmeticException {

        int int1 =3, int2=0;
    try{
        quotient (int1, int2);
    }   finally{
        System.out.println("Exception: an integer cannot be devided by zero")
    }

    }

    public static int quotient(int num1, int num2){
        if (num2 == 0){
           throw new ArithmeticException("you cannot devide by zero");
        }
        return num1/num2;
```

AFTER THIS WEEK YOU SHOULD KNOW:

- Exception handling
- Try-catch-finally statements
- System exceptions
- Runtime exceptions
- Throws
- Throw