JAVA PROGRAMMING

20 MCA 2 2 C

Unit – III

EXCEPTION HANDLING

FACULTY

Dr. K. ARTHI MCA, M.Phil., Ph.D.,

Assistant Professor,
Postgraduate Department of Computer Applications,
Government Arts College (Autonomous),
Coimbatore – 641 018.

Exception Handling in Java

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

In this page, we will learn about Java exceptions, its type and the difference between checked and unchecked exceptions.

What is Exception in Java

Dictionary Meaning: Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantage of Exception Handling

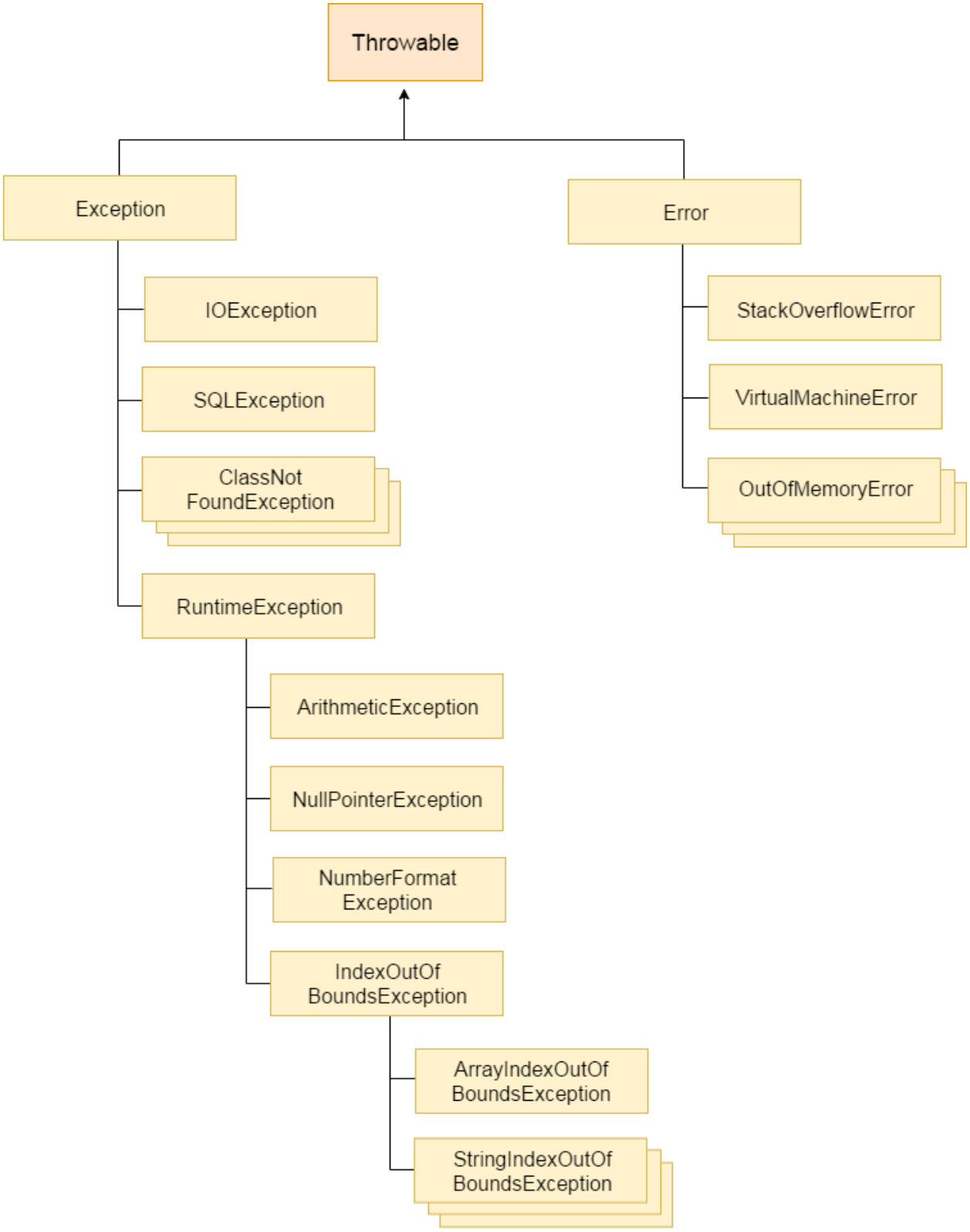
The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

```
1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;
```

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compiletime, but they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

---

next →← prev

Exception Handling in Java

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

In this page, we will learn about Java exceptions, its type and the difference between checked and unchecked exceptions.

What is Exception in Java

Dictionary Meaning: Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

. statement 1;
. statement 2;
. statement 3;
. statement 4;
. statement 5;//exception occurs
. statement 6;
. statement 7;
. statement 8;
. statement 9;
0. statement 10;

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Do You Know?

- What is the difference between checked and unchecked exceptions?
- What happens behind the code int data=50/0;? ○ Why use multiple catch block? ○ Is there any possibility when finally block is not executed?
- What is exception propagation?
- What is the difference between throw and throws keyword?
- What are the 4 rules for using exception handling with method overriding?

Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException,

ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Java Exception Handling Example

Let's see an example of Java Exception Handling where we using a try-catch statement to handle the exception.

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

.   //code that may raise exception
.   int data=100/0;
. }catch(ArithmeticException e){System.out.println(e);}
. //rest code of the program
. System.out.println("rest of the code...");
```

Test it Now

Output:

```
Exception in thread main
java.lang.ArithmeticException:/ by zero
rest of the code...
```

In the above example, 100/0 raises an ArithmeticException which is handled by a try-catch block.

```
. try{

. }
0. }
```

Common Scenarios of Java Exceptions

There are given some scenarios where unchecked exceptions may occur. They are as follows:

1) A scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

```
1. int a=50/0;//ArithmeticException
```

---

2) A scenario where NullPointerException occurs

If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

```
1. String s=null;
2. System.out.println(s.length());//NullPointerException
```

---

3) A scenario where NumberFormatException occurs

The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.

```
1. String s="abc";
2. int i=Integer.parseInt(s);//NumberFormatException
```

---

4) A scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

```
1. int a[]=new int[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException
```

Multithreading in Java

1. Multithreading
2. Multitasking
3. Process-based multitasking
4. Thread-based multitasking
5. What is Thread

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Creating a thread

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.

8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).

16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

Starting a thread:

start() method of Thread class is used to start a newly created thread. It performs following tasks:

○ A new thread starts(with new callstack). ○ The thread moves from New state to the Runnable state. ○ When the thread gets a chance to execute, its target run() method will run.

1) Java Thread Example by extending Thread class

```
class Multi extends Thread{  public
void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();   t1.start();
 }
}
```

```
Output:thread is running...
```

2) Java Thread Example by implementing Runnable interface

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
Output:thread is running...
```

Synchronization in Java

Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.
    3. static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block
3. by static synchronization

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

From Java 5 the package java.util.concurrent.locks contains several lock implementations.

Understanding the problem without Synchronization

In this example, there is no synchronization, so output is inconsistent. Let's see the example:

```
1.      class Table{
2.      void printTable(int n){//method not synchronized
3.      for(int i=1;i<=5;i++){
4.      System.out.println(n*i);
5.      try{
6.      Thread.sleep(400);
7.      }catch(Exception e){System.out.println(e);}
8.      }
9.
10.  }
11.  }
12.
13. class MyThread1 extends Thread{
14. Table t;
15. MyThread1(Table t){
16. this.t=t;
17. }
18. public void run(){
19. t.printTable(5);
20. }
21.
22. }
23. class MyThread2 extends Thread{
24. Table t;
25. MyThread2(Table t){
26. this.t=t;
27. }
28. public void run(){
29. t.printTable(100);
30. }
31. }
32.
33. class TestSynchronization1{
34. public static void main(String args[]){
35. Table obj = new Table();//only one object
36. MyThread1 t1=new MyThread1(obj);
37. MyThread2 t2=new MyThread2(obj);
38. t1.start();
39. t2.start();
40. }
```

```
41. }
Output: 5
        100
        10
        200
        15
        300
        20
        400
        25
        500
```

Java synchronized method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
1.      //example of java synchronized method
2.      class Table{
3.      synchronized void printTable(int n){//synchronized method
4.      for(int i=1;i<=5;i++){
5.      System.out.println(n*i);
6.      try{
7.      Thread.sleep(400);
8.      }catch(Exception e){System.out.println(e);}
9.      }
10.
11.  }
12.  }
13.
14. class MyThread1 extends Thread{
15. Table t;
16. MyThread1(Table t){
17. this.t=t;
18. }
19. public void run(){
20. t.printTable(5);
21. }
22.
23. }
24. class MyThread2 extends Thread{
25. Table t;
26. MyThread2(Table t){
27. this.t=t;
28. }
29. public void run(){
30. t.printTable(100);
31. }
32. }
33.
34. public class TestSynchronization2{
35. public static void main(String args[]){
36. Table obj = new Table();//only one object
37. MyThread1 t1=new MyThread1(obj);
38. MyThread2 t2=new MyThread2(obj);
```

```
39. t1.start();
40. t2.start();
41. }
42. }
```

```
Output: 5
        10
        15
        20
        25
        100
        200
        300
        400
        500
```

Inter-thread communication in Java

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

- wait() ○ notify()
- notifyAll()

---

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description
public final void wait()throws InterruptedException	waits until object is notified.
public final void wait(long timeout)throws InterruptedException	waits for the specified amount of time.

---

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax: public final void notify()

---

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor. Syntax: public final void

notifyAll()

Understanding the process of inter-thread communication

The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

Why wait(), notify() and notifyAll() methods are defined in Object class not Thread class?

It is because they are related to lock and object has a lock.

Difference between wait and sleep?

Let's see the important differences between wait and sleep methods.

wait()	sleep()
wait() method releases the lock	sleep() method doesn't release the lock.
is the method of Object class	is the method of Thread class
is the non-static method	is the static method
is the non-static method	is the static method
should be notified by notify() or notifyAll() methods	after the specified amount of time, sleep is completed.

Example of inter thread communication in java

Let's see the simple example of inter thread communication.

```
1. class Customer{  2. int
amount=10000;
3.
4.  synchronized void withdraw(int amount){
5.  System.out.println("going to withdraw...");
6.
7.  if(this.amount<amount){
8.  System.out.println("Less balance; waiting for deposit...");
9.  try{wait();}catch(Exception e){}
10. }
11. this.amount-=amount;
12. System.out.println("withdraw completed...");
13. }
14.
15. synchronized void deposit(int amount){
16. System.out.println("going to deposit...");
17. this.amount+=amount;
18. System.out.println("deposit completed... ");
19. notify();
20. }
21. }
22.
23. class Test{
24. public static void main(String args[]){
25. final Customer c=new Customer();
26. new Thread(){
27. public void run(){c.withdraw(15000);}
28. }.start();
29. new Thread(){
30. public void run(){c.deposit(10000);}
31. }.start();
32.
33. }}
```

```
Output: going to withdraw...        Less balance;
waiting for deposit...       going to deposit...
deposit completed...        withdraw completed
```