Exception H

andBug

Exception

Exception

□ An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

□ A user has entered invalid data.

□ A file that needs to be opened cannot be found. □ A network connection has been lost in the middle of communications, or the JVM has run out of memory. □ Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Categories of

Categories of

Exception

Exception

- **Checked exceptions:** A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Exception Hierarchy

Exception Hierarchy

- All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.
- Errors are not normally trapped form the Java programs. These conditions normally happen in case of severe failures, which are not handled by the java programs. Errors are generated to indicate errors generated by the runtime environment.
- **Example**: JVM is out of Memory. Normally programs cannot recover from errors.
- The Exception class has two main subclasses :
  - IOException class
  - RuntimeException Class.

Catching Exceptions

Catching Exceptions

- A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
{
//Protected code
}catch(ExceptionName e1)
{
 //Catch block
}
```

- A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Example:

```
// File Name : ExcepTest.java
import java.io.*;
public class ExcepTest{ public static
void main(String args[]) {
Try
{
int a[] = new int[2];
System.out.println("Access element
three :" + a[3]);  }
catch(ArrayIndexOutOfBoundsExceptio
n e)
{
System.out.println("Exception thrown
:" + e);  }
System.out.println("Out of the block");
}
}
```

Syntax:

Multiple Catch

Multiple Catch Blocks

```
try
{
//Protected code
}
catch(ExceptionType1 e1)
{
//Catch block
}
catch(ExceptionType2 e2)
{
//Catch block
}
catch(ExceptionType3 e3)
{
//Catch block
}
```

Example:

```
try
{
   file = new FileInputStream(fileName);
  x = (byte) file.read();
}
catch(IOException i)
{
i.printStackTrace();
return -1;
}
```

```
 catch(FileNotFoundException f) //Not valid!
{
f.printStackTrace();
return -1;
}
```

Throws/Throw

Throws/Throw

•If a method does not handle a checked exception, the method must declare it using the throwskeyword.

•The throws keyword appears at the end of a method's signature.

•You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the throw keyword.

Remote Exception

Remote Exception

```
import java.io.*;
public class className
{
  public void deposit(double amount) throws
    RemoteException {
      // Method implementation
      throw new RemoteException();
  }
  //Remainder of class definition
}
```

Example:

Function throws a RemoteException and an InsufficientFundsException

```
import java.io.*;
public class className
{
  public void withdraw(double amount) throws RemoteException,
                                             InsufficientFundsExceptio
                                             n
    {
      // Method implementation
    }
    //Remainder of class definition
}
```

Finally Keyword

- The finally keyword is used to create a block of code that follows a try block.

- A finally block of code always executes, whether or not an exception has occurred.

- Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

Syntax:

```
try
{
  //Protected code
}
catch(ExceptionType1 e1)
{
  //Catch block
}
catch(ExceptionType2 e2)
{
  //Catch block
}
catch(ExceptionType3 e3)
{
  //Catch block
}
finally
{
   //The finally block always executes.
}
```

Example:

```
public class ExcepTest
```

```
{
  public static void main(String args[])
  {
    int a[] = new int[2];
    try
    {
       System.out.println("Access element three :" + a[3]);
    }
    catch(ArrayIndexOutOfBoundsException e)
    {
      System.out.println("Exception thrown :" + e);
    }
    finally
    {
      a[0] = 6;
      System.out.println("First element value: " +a[0]);
      System.out.println("The finally statement is executed");
    }
  }
}
```

Declaring Your Own Exception

✔You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes:

- € All exceptions must be a child of Throwable.
- € If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- € If you want to write a runtime exception, you need to extend the RuntimeException class.

Syntax:

```
class MyException extends Exception{ }
```

Example Module:

```
// File Name InsufficientFundsException.java
import java.io.*;
public class InsufficientFundsException extends Exception
{
  private double amount;
  public InsufficientFundsException(double amount)
  {
    this.amount = amount;
  }
  public double getAmount()
  {
    return amount;
  }
}
```

Example Program:

```
// File Name CheckingAccount.java
import java.io.*;

public class CheckingAccount
{
  private double balance;
  private int number;
  public CheckingAccount(int number)
  {
     this.number = number;
  }
  public void deposit(double amount)
  {
     balance += amount;
  }
public void withdraw(double amount) throws
InsufficientFundsException {
   if(amount <= balance)
   {
      balance -= amount;
   }
```

```
        else
        {
           double needs = amount - balance;
           throw new InsufficientFundsException(needs);
        }
     }
     public double getBalance()
     {
         return balance;
     }
     public int getNumber()
     {
         return number;
     }
   }
// File Name BankDemo.java
public class BankDemo
{
public static void main(String [] args)
{
CheckingAccount c = new
CheckingAccount(101);
System.out.println("Depositing $500...");
c.deposit(500.00);
try
{
System.out.println("\nWithdrawing
$100...");  c.withdraw(100.00);
System.out.println("\nWithdrawing
$600...");  c.withdraw(600.00);
}
```

```
catch(InsufficientFundsException e)
{
System.out.println("Sorry, but you are
short $" +  e.getAmount());
e.printStackTrace();
```

```
}
}
}
```

deposit() and withdraw() methods

Other Exception

It is possible to define two categories of Exceptions and Errors.

- **JVM Exceptions:** These are exceptions/errors that are exclusively or logically thrown by the JVM.
  - Examples: NullPointerException, ArrayIndexOutOfBoundsException, ClassCastException.

- **Programmatic Exceptions:** These exceptions are thrown explicitly by the application or the API programmers Examples: IllegalArgumentException, IllegalStateException