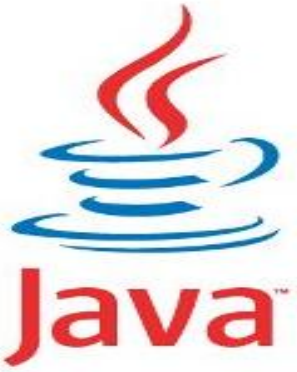

JAVA SE

(CORE JAVA)

LECTURE-28

Today's Agenda

- Obtaining description of an Exception
- Using the keyword throw
- Checked and Unchecked Exception

Obtaining description of an Exception

- Whenever an exception occurs in try block, java creates an object of a specific Exception class and stores some important details in the object.
- Now using that object we can obtain all the possible details of the exception that occurred in the catch block.
- To do this we can use several methods using the object reference. Some important and common methods are

Methods

- **public String getMessage( ):-**
- This method is present in the class **Throwable**.
- The method returns “error message” generated by java regarding the exception.
- Example:- catch(Exception e)
  {
  System.out.println(**e.getMessage()**);
  }
- **public String toString( ):-**
- This method is present in the Object class hence, in all classes.
- It is invoked in two situations

1. In method println( ).
2. Concatenation of object and string.

Overriding the "toString( )" Method

- Example:-

```
class Person
{
 private int age;
 private String name;
 public Person(int age, String name)
 {
 this.age=age;
 this.name=name;
 }
}
class UsePerson
{
public static void main(String [ ]
    args)
{
 Person p=new Person(21,"Amit");
 System.out.println(p);
 }
}
```

```
F:\Java Codes>java UsePerson
Person@15db9742
```

This is called Hashcode.

Overriding the "toString( )" Method

- A hashcode is an unique number generated from any object. This is what allows objects to be stored/retrieved quickly.
- But it is of no use to programmer or user.
- On passing object reference to println() method, toString() is invoked. Since, we have not overridden it so the base class version gets called and hence we get hashcode in return.
- So, by overriding the toString() method we can get information related to a specific class and not hashcode.

Overriding the "toString( )" Method

```
class Person
{
 private int age;
 private String name;
 public Person(int age, String name)
 {
 this.age=age;
 this.name=name;
 }
 public String toString()
 {
 return name+", "+age;
 }
}
```

```
class UsePerson
{
public static void main(String [ ]
     args)
{
 Person p=new Person(21,"Amit");
 System.out.println(p);
 }
}
```

```
F:\Java Codes>java UsePerson
Amit, 21
```

- The exception classes have also overridden toString() method, it returns the name of exception along with error message for an exception generated by java.

- The toString() of specific exception class is called as per the object formed in the try block.

Using “throw” keyword

- In java, exceptions occur on JVM’s choice i.e. anything against java’s flow. Example, if denominator is 0 then java itself generates ArithmeticException.

- But, in some situation a programmers may want to generate or define exceptions on their choice.

- To do this, java provides us the keyword throw. It’s syntax is

throw <Some exception class object reference>;

Let us understand this through an example.

Example

- WAP to accept two integers from user. If the denominator entered is 0 then show java's exception message and if the numerator entered is 0 then generate your own exception displayed "Numerator must be positive".
- Sample Output:-

```
F:\Java Codes>java Test
Enter two integers
10
0
/ by zero

F:\Java Codes>java Test
Enter two integers
0
25
Numerator should be positive
```

Example


```
import java.util.Scanner;
class Test
{
    public static void main(String [] args)
    {
        Scanner kb=new Scanner(System.in);
        try
        {
        System.out.println("Enter two integers");
        int a=kb.nextInt();
        int b=kb.nextInt();
            if(a<=0)
            {
                ArithmeticException obj=new ArithmeticException("Numerator should be positive");
                throw obj;
            }
            int c=a/b;
            System.out.println("Division is "+c);
        }
        catch(ArithmeticException ex)
        {
            System.out.println(ex.getMessage());
        }
    }
}
```

Classification of Exceptions

Checked Exceptions

Exceptions for which java forces the programmer to either handle it using try-catch or the programmer must inform or warn the caller of the method that his code is not handling the checked exception.
The warning given is through the keyword **throws**. It is used in the method 's prototype along with exception class name.

Unchecked Exceptions

Exceptions for which java never compels the programmer to handle it.
The program would be compiled and executed by Java.
Example the class RuntimeException and all its derived classes fall in this category.

Checked Exception

- This in java is called handle of declare rule.
- The caller also has 2 options, either use try-catch or the keyword throws.
- If every method uses throws, then ultimately the responsibility goes to the JVM which will follow the standard mechanism of handling the exception.
- All exceptions which are not derived classes of RuntimeException fall in this category. Like SQLException, IOException, InterruptedException etc…

Code for Checked Exception

```
import java.io.*;
class Input
{
 public static void accept( ) throws IOException
 {
 System.out.println("Enter a character");
 char c=(char)System.in.read( );
 System.out.println("You entered "+c);
 }
}
class UseInput
{
 public static void main(String [] args) throws IOException
 {
 Input.accept( );
 }
}
```

End Of Lecture 28

Thank You

For any queries mail us @: scalive4u@gmail.com
Call us @ : 0755-4271659, 7879165533

Agenda for Next Lecture:

1. **Creating programmer defined\Custom Exceptions**