Java - Custom Exception

Java Custom Exception

The custom exception refers to the creation of your own exception to customize an exception according to the needs. The custom exceptions are derived from the **`Exception`** class.

Need of Java Custom Exceptions

- To categorize the exceptions based on the different types of errors in your project.
- To allow application-level exception handling.

Create a Custom Exception in Java

To create a custom exception, you need to create a class that must be inherited from the Exception class.

Syntax

Here is the syntax to create a custom class in Java -

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

You just need to extend the predefined **Exception** class to create your own Exception. These are considered to be checked exceptions.

Rules to Create Custom Exception

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.

Java Custom Exception Example

The following **InsufficientFundsException** class is a user-defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.

```
class InsufficientFundsException extends Exception {
   private double amount;

   public InsufficientFundsException(double amount) {
      this.amount = amount;
   }

   public double getAmount() {
      return amount;
   }
}
```

To demonstrate using our user-defined exception, the following CheckingAccount class contains a withdraw() method that throws an InsufficientFundsException.

```
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;
      }
   }
```

The following BankDemo program demonstrates invoking the deposit() and withdraw() methods of CheckingAccount.

</> Open Compiler

```
package com.tutorialspoint;

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();
      }
   }
}

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;
   }
}

class InsufficientFundsException extends Exception {
   private double amount;

   public InsufficientFundsException(double amount) {
      this.amount = amount;
   }

   public double getAmount() {
      return amount;
   }
}
```

Output

Compile all the above three files and run BankDemo. This will produce the following result –

```
Depositing $500...

Withdrawing $100...

Withdrawing $600...
Sorry, but you are short $200.0
com.tutorialspoint.InsufficientFundsException
        at com.tutorialspoint.CheckingAccount.withdraw(BankDemo.java:39)
        at com.tutorialspoint.BankDemo.main(BankDemo.java:14)
```

In next example, we're declaring our custom exception as RuntimeException to make it as unchecked exception class as below –

```
class MyException extends RuntimeException {
}
```

Example to Create Custom Class by Extending Runtime Exception

We are extending the predefined **RuntimeException** class to create your own Exception as an unchecked exception. The following **InsufficientFundsException** class is a user-defined exception that extends the RuntimeException class, making it a unchecked exception. An RuntimeException class is like any other class, containing useful fields and methods.

```
class InsufficientFundsException extends RuntimeException {
   private double amount;

   public InsufficientFundsException(double amount) {
      this.amount = amount;
   }

   public double getAmount() {
      return amount;
   }
}
```

The following BankDemo program demonstrates invoking the deposit() and withdraw() methods of CheckingAccount using unchecked exception.

</>
Open Compiler

```
package com.tutorialspoint;

public class BankDemo {

   public static void main(String [] args) {
      CheckingAccount c = new CheckingAccount(101);
      System.out.println("Depositing $500...");
      c.deposit(500.00);

      System.out.println("\nWithdrawing $100...");
      c.withdraw(100.00);
      System.out.println("\nWithdrawing $600...");
      c.withdraw(600.00);

   }
}

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) {
      if(amount <= balance) {
         balance -= amount;
      }else {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }

   public double getBalance() {
```

```
      return balance;
   }

   public int getNumber() {
      return number;
   }
}

class InsufficientFundsException extends RuntimeException {
   private double amount;

   public InsufficientFundsException(double amount) {
      this.amount = amount;
   }

   public double getAmount() {
      return amount;
   }
}
```

Output

Compile and run BankDemo. This will produce the following result –

```
Depositing $500...

Withdrawing $100...

Withdrawing $600...
Exception in thread "main"
com.tutorialspoint.InsufficientFundsException
      at com.tutorialspoint.CheckingAccount.withdraw(BankDemo.java:35)
      at com.tutorialspoint.BankDemo.main(BankDemo.java:13)
```