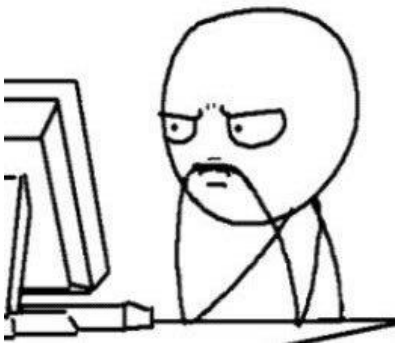
Custom Exceptions in Java

If you are **creating your own Exception** that is known as custom exception. Java custom exceptions are used to customize the exception according to user need.

Before knowing how to create custom exception let us see how java cannot make out some exceptions.

Example: Let us consider atm example



```
import java.util.Scanner;
class ATM
{
    int acc_num=1234;
    int password=9999;
    int an,pwd;
    void acceptInput()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the acc_num");
        an = scan.nextInt();
        System.out.println("Enter the password");
        pwd = scan.nextInt();
    }
    void verify()
    {
        if(acc_num==an && password==pwd)
        {
            System.out.println("Colletc your money");
        }
        else
        {
            System.out.println("Invalid card details. Try again!!");
        }
    }
}
class Bank
{
    void initiate()
    {
        ATM atm = new ATM();
        atm.acceptInput();
        atm.verify();
    }
}
class Demo
{
    public static void main(String[] args)
    {
        Bank b = new Bank();
        b.initiate();
    }
}
```

Output:

```
Enter the acc_num
1234
Enter the password
9999
Colletc your money
Press any key to continue . . .
```

```
Enter the acc_num
1234
Enter the password
8888
Invalid card details. Try again!!
Press any key to continue . . .
```

In ideal atm case if the entered password/pin is not correct then we get two more chances but here that is not happening... So this can be overcome by using custom exception which next we'll be sseing.

Java provides us facility to create our own exceptions which are basically derived classes of Exception.

Example using custom exception:


```
import java.util.Scanner;
class InvalidUserException extends Exception
{
    public String getMessage()
    {
        return("Invalid card details. Try again!!");
    }
}
class ATM
{
    int acc_num=1234;
    int password=9999;
    int an,pwd;
    void acceptInput()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the acc_num");
        an = scan.nextInt();
        System.out.println("Enter the password");
        pwd = scan.nextInt();
    }
    void verify() throws Exception
    {
        if(acc_num==an && password==pwd)
        {
            System.out.println("Colletc your money");
        }
        else
        {
            InvalidUserException iue = new InvalidUserException();
            System.out.println(iue.getMessage());
            throw iue;
        }
    }
}
class Bank
{
    void initiate()
    {
        ATM atm = new ATM();
        try
        {
            atm.acceptInput();//first attempt
            atm.verify();
        }
        catch (Exception e)
        {
            try
            {
                atm.acceptInput();//second attempt
                atm.verify();
            }
```

```
            catch (Exception f)
            {
                try
                {
                    atm.acceptInput();//third and final attempt
                    atm.verify();
                }
                catch (Exception g)
                {
                    System.out.println("Card blocked!!");
                    System.exit(0);
                }
            }
        }
    }
}
class Demo
{
    public static void main(String[] args)
    {
        Bank b = new Bank();
        b.initiate();
    }
}
```

Output for invalid enteries:

```
Enter the acc_num
1234
Enter the password
7777
Invalid card details. Try again!!
Enter the acc_num
1234
Enter the password
3333
Invalid card details. Try again!!
Enter the acc_num
1234
Enter the password
5555
Invalid card details. Try again!!
Card blocked!!
Press any key to continue . . .
```

- **Create a class and extend the Exception class.**
- **Override the getMessage() and provide a suitable message.**
- **Explicitly create an object of the class wherever there is need for an exception to be generated.**

If you are wondering why or where exactly in real life the custom exception are being used then let us see that:

Java exceptions cover almost all general exceptions that are bound to happen in programming.

However, we sometimes need to supplement these standard exceptions with our own.

The main reasons for introducing custom exceptions are:

- **Business logic exceptions** – Exceptions that are specific to the business logic and workflow. These help the application users or the developers understand what the exact problem is.
- To catch and provide specific treatment to a subset of existing Java exceptions.
- **Custom exceptions are very much useful when we need to handle specific exceptions related to the business logic**. When used properly, they can serve as a useful tool for better exception handling and logging.